ON THE DYNAMICS OF HUMAN LOCOMOTION AND CO-DESIGN OF LOWER LIMB ASSISTIVE DEVICES

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the service academique.

Thèse n. XXX 2014
présenté le 14 Mai 2014
à la Faculté des Sciences de Base
laboratoire Biorobotics
programme doctoral Systèmes de production et robotique
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Jesse van den Kieboom

acceptée sur proposition du jury :

Prof José del R. Millán, président du jury
Prof Auke Jan Ijspeert, directeur de thèse
Prof Joshua C. Bongard, rapporteur
Prof Silvestro Micera, rapporteur
Prof Hartmut Geyer, rapporteur

Lausanne, EPFL, 2014

The most exciting phrase to hear in science,
the one that heralds new discoveries,
is not ’Eureka !’, but
’That’s funny...’
— Isaac Asimov

Acknowledgements

First of all, I would like to thank my supervisor, Auke Jan Ijspeert, for giving me the opportunity to do the work presented here in this thesis at Biorob EPFL. Without his continuous support, advice and guidance there would not have been a thesis to read. I also want to thank all the people involved in the EVRYON European project for providing a productive and most importantly, collaborative environment which helped bring together expertise from many fields. In particular, I want to thank Renaud Ronsse, which has provided invaluable insights and the occasional Dutch greeting, during his post doctoral position at Biorob. Besides every member of the group of great people at Biorob, providing an excellent, productive yet fun atmosphere, special thanks go to Jérémie Knüsel who has been sitting across of me for more than 4 years and has provided a much needed sounding board at more occasions than I can count. I would also like to thank my family who have supported me unconditionally throughout the years. Finally, a very special person who deserves most of my gratitude is Fanny, who has stood by me for all these years whether the good, the bad or the ugly, thank you.

Lausanne, 14 May 2014 J.K.

Abstract

Recent developments in lower extremities wearable robotic devices for the assistance and rehabilitation of humans suffering from an impairment have led to several successes in the assistance of people who as a result regained a certain form of locomotive capability. Such devices are conventionally designed to be anthropomorphic. They follow the morphology of the human lower limbs. It has been shown previously that non-anthropomorphic designs can lead to increased comfort and better dynamical properties due to the fact that there is more morphological freedom in the design parameters of such a device. At the same time, exploitation of this freedom is not always intuitive and can be difficult to incorporate. In this work we strive towards a methodology aiding in the design of possible non-anthropomorphic structures for the task of human locomotion assistance by means of simulation and optimization. The simulation of such systems requires state of the art rigid body dynamics, contact dynamics and, importantly, closed loop dynamics. Through the course of our work, we first develop a novel, open and freely available, state of the art framework for the modeling and simulation of general coupled dynamical systems and show how such a framework enables the modeling of systems in a novel way. The resultant simulation environment is suitable for the evaluation of structural designs, with a specific focus on locomotion and wearable robots. To enable open-ended co-design of morphology and control, we employ population-based optimization methods to develop a novel Particle Swarm Optimization derivative specifically designed for the simultaneous optimization of solution structures (such as mechanical designs) as well as their continuous parameters. The optimizations that we aim to perform require large numbers of simulations to accommodate them and we develop another open and general framework to aid in large scale, population based optimizations in multi-user environments. Using the developed tools, we first explore the occurrence and underlying principles of natural human gait and apply our findings to the optimization of a bipedal gait of a humanoid robotic platform. Finally, we apply our developed methods to the co-design of a non-anthropomorphic, lower extremities, wearable robot in simulation, leading to an iterative co-design methodology aiding in the exploration of otherwise hard to realize morphological designs.

Keywords : dynamical systems, rigid body dynamics, bipedal locomotion, natural gait, robotics, optimization, impedance control, wearable robot co-design, morphology

Résumé

Les récents développements de dispositifs robotiques portables pour les membres inférieurs, utilisés pour l'assistance ainsi que pour la réhabilitation de personnes souffrant de détérioration des fonctions motrices, a permis la récupération de certaines capacités locomotrices. Ces dispositifs sont conventionnellement conçus pour être anthropomorphique, et suivent la morphologie des membres inférieures humains. Il a cependant été montré que des modèles non-anthropomorphiques, permettant plus de liberté quant au choix de conception et du nombre de paramètres, pourraient offrir un meilleur confort et de meilleures propriétés dynamiques. Cependant gérer et exploiter cette liberté de conception n'est pas intuitive et peut être difficile à gérer. Dans cette étude, nous présentons une méthodologie basée sur des simulations physiques combinées à des optimisations mathématiques pouvant aider la conception de structures robotiques non-anthropomorphiques dédiées à l'assistance de la marche humaine. La simulation de ces structures nécessite l'utilisation des avancées les plus récentes en dynamique des corps rigides, des contacts physiques ainsi que des systèmes rétroactifs. Nous avons tout d'abord développé un nouvel outil, ouvert et disponible gratuitement, permettant la modélisation et la simulation des systèmes dynamiques couplées, et proposant une nouvelle manière de modéliser ces systèmes. L'environnement de simulation résultant peut être utilisé pour l'évaluation de conceptions structurelles, en particulier dans le cadre de la marche et les dispositifs robotiques portables. Pour permettre une conception mixte de la morphologie et du contrôle, nous avons développé un nouveau dérivé de l'optimisation par essaims particulaires, spécifiquement conçu pour l'optimisation simultanée de structures (par exemple une structure mécanique), ainsi que les paramètres continus associés. L'optimisation que nous souhaitons réaliser requérant un grand nombre de simulation, nous avons donc développé un autre outil ouvert et accessible librement, permettant le déploiement d'optimisation à grande échelle dans un environnement multi-utilisateurs. En utilisant les outils développés, nous avons ensuite exploré les principes fondamentaux de la marche humaine et appliqué nos découvertes à l'optimisation d'une démarche bipède d'un robot humanoïde. Enfin, nous avons combinées les méthodes et outils développés pour le co-design, et les avons appliqués à la simulation d'un dispositif robotique non-anthropomorphique des membres inférieures humains. Cette démarche a conduit à une méthodologie de co-design itérative permettant l'exploration de concepts morphologiques autrement difficile à réaliser.

Mots clefs : systèmes dynamique, la dynamique des corps rigides, marche bipède, démarche naturelle, robotique, optimisation, contrôle en impédance, co-design de dispositif robotique

Contents

Acknowledgements **v**

Abstract **vii**

List of figures **xxi**

List of tables **xxiii**

List of models **xxv**

1 Introduction **1**
  1.1 Co-design of human assistive devices 1
  1.2 Human locomotion 3
  1.3 Tools 3
  1.4 Organization of the thesis 5

I Dynamics and optimization **9**

2 Dynamics **11**
  2.1 cȯdγn, coupled dynamics 12
  2.2 Core concepts 13
    2.2.1 Nodes and edges 13
    2.2.2 Mathematical language 15
    2.2.3 Edge projections 21
    2.2.4 Events 21
    2.2.5 Numerical integration 22
  2.3 Modeling language 23
    2.3.1 Variables and simple differential equations 24
    2.3.2 Nodes 26
    2.3.3 Edges 28
    2.3.4 Generators and selectors 28
    2.3.5 Templates 33
    2.3.6 Integrator 33
    2.3.7 Events 34

2.4 Example I: Central pattern generators . . . 37
  2.4.1 Van der Pol . . . 38
  2.4.2 Matsuoka . . . 40
  2.4.3 Morphed nonlinear phase oscillator . . . 40
2.5 Example II: SLIP model . . . 42
2.6 Rigid body dynamics . . . 45
  2.6.1 Existing simulators . . . 47
  2.6.2 Deriving equations of motion . . . 50
  2.6.3 Spatial vector algebra . . . 51
  2.6.4 Joint models . . . 55
  2.6.5 Model definition . . . 59
  2.6.6 Inverse dynamics . . . 59
  2.6.7 Forward dynamics . . . 61
  2.6.8 Jacobian . . . 62
  2.6.9 Closed loop dynamics . . . 64
  2.6.10 Contact modeling . . . 69
  2.6.11 Visualization . . . 73
2.7 Performance . . . 73
  2.7.1 libcodyn . . . 73
  2.7.2 The road towards performance . . . 75
  2.7.3 As *raw* as *C* . . . 76
  2.7.4 Performance comparison . . . 79
2.8 Tools . . . 81
  2.8.1 Command line tools . . . 82
  2.8.2 Graphical designer interface . . . 83
  2.8.3 Supported languages . . . 83
2.9 Availability . . . 84
2.10 Conclusion . . . 85

3 Optimization **89**
3.1 Population-based methods . . . 90
  3.1.1 Genetic Algorithms . . . 91
  3.1.2 Genetic Programming . . . 91
  3.1.3 Particle swarm optimization . . . 93
3.2 Metamorphic particle swarm optimization . . . 95
  3.2.1 Metamorphic PSO Algorithm . . . 96
  3.2.2 Properties . . . 101
  3.2.3 Applications . . . 105
  3.2.4 Discussion . . . 106
3.3 Multi objective particle swarm optimization . . . 107
  3.3.1 Multi objective optimization . . . 107
  3.3.2 Multi objective PSO using lexicographic ordering . . . 108

3.4 Large scale population-based optimization 112
  3.4.1 Conceptual overview 112
  3.4.2 User layer 112
  3.4.3 Server layer 114
  3.4.4 Workstation layer 115
  3.4.5 Results and analysis 115
  3.4.6 Availability 115
  3.4.7 In the wild 116
3.5 Conclusion 116

II Human locomotion and assistance **119**

4 Optimization of natural human gait **121**
4.1 Human gait optimization 122
  4.1.1 Model 123
  4.1.2 Optimization 126
  4.1.3 Results 129
  4.1.4 Discussion 132
4.2 CoMan humanoid robot gait optimization 133
  4.2.1 CoMan humanoid robot platform 134
  4.2.2 Modeling the CoMan robot 135
  4.2.3 Study I: CoMan gait optimization 137
  4.2.4 Study II: The effect of impedance control during perturbations 142
4.3 Conclusion 145

5 Co-design of human assistive devices **149**
5.1 Initial study 152
5.2 Design 154
  5.2.1 Topologies and morphologies 155
  5.2.2 Actuator placement 157
  5.2.3 Singularities 159
  5.2.4 Summary 161
5.3 Modeling 162
  5.3.1 Augmented human model 162
  5.3.2 Closing joint actuation 163
  5.3.3 Constraint forces in closed loop systems 163
  5.3.4 Control 164
5.4 Optimization 165
  5.4.1 Algorithm 165
  5.4.2 Parameterization 165
  5.4.3 Objectives 167
  5.4.4 Singularities 168

5.5 Results 171
5.6 Discussion 174

6 Conclusion **177**

A Models **181**

B Complementary figures **185**

Bibliography **199**

Curriculum Vitae **201**

List of Figures

2.1 A conceptual representation of a network of two nodes which are bidirectionally coupled. . . . 15

2.2 Left: System output of a simple point mass accelerating due to gravity. Right: conceptual representation of a self-edge 26

2.3 System output of two simple point masses accelerating due to gravity and simple air friction. 27

2.4 System output of two simple point mass accelerating due to gravity and undergoing forces from air friction. Additionally, the two point masses are coupled by a bidirectional spring. 28

2.5 Graphical representation of a network of two coupled point masses. 29

2.6 System output of a densely coupled system of point masses. 32

2.7 Conceptual rendering of a network of a densely coupled system of point masses. The simple model definition in model 2.5 using generators demonstrates the generative abilities of the cȯdγn language. 32

2.8 Output of multiple pogo-point mass dynamics with various initial conditions, masses and spring/damper characteristics. 37

2.9 Output of a basic system of phase coupled Hopf oscillators. The 5 oscillators are initially not phase locked, but quickly converge to their desired phase locked behavior 40

2.10 Output of a coupled system of two morphed oscillators with arbitrary shaping functions. The two oscillators start outside their limit cycle but quickly converge. At the same time, a bidirectional coupling on the phase ensures phase locking behavior after a short period of time. 43

2.11 Schematic depiction of the SLIP model. The basic SLIP model consists of a single point mass m connected to a massless spring with rest length l. The angle of attack α determines at which angle the leg transitions from the swing phase to the stance phase during locomotion. 43

2.12 Number of steps to fall for the SLIP model for different combinations of spring stiffness and angle of attack using cȯdγn. The characteristic J-shape indicates the stable region of forward locomotion (Seyfarth *et al.*, 2002). 45

2.13 System output of a rigid body dynamics simulation of a chain of 5 pendulums. The first pendulum starts out at an angle and starts accelerating due to gravity. The system is damped by a simple velocity based damping term on the generalized forces of each of the pendulum joints. 59

2.14 Left: a schematic representation of a closed loop system of one degree. The original system consists of 3 revolute joints and is closed to the fixed base through p_4. Right: Output of the simulated system. The resulting joint angle motions are consistent with the imposed constraint and can be described by a single variable. 70

2.15 Left: a schematic representation of the leg. The open loop system consists of 4 revolute joints, hip, knee, ankle and par. A closing joint, cl, creates a pantographic structure having only 2 degrees of freedom. Right: output of simulating the system. A damping force acts on the hip and a spring force acts on the knee, causing quick oscillations. 70

2.16 Screenshot of cȯdyn integration in blender. It shows a rendering of the closed loop system presented in model 2.21. Models can be directly imported from cȯdyn files and simulated using the Blender game engine. 74

2.17 Simulation performance of Bullet, SimMechanics, codyn and rawc on systems with increasing number of degrees of freedom. Simulations were performed for 30 simulated seconds. The y-axis shows in logarithmic scale the real time it took for the simulations to finish. As expected, codyn itself is an order of magnitude slower than both Bullet and SimMechanics (which show similar performance). When using rawc to generate optimized code, we however obtain another order of magnitude faster simulations, for systems up to 20 degrees of freedom than Bullet or SimMechanics. Both Bullet and SimMechanics show better scaling properties towards systems with a larger number of DOFs. 81

2.18 Performance of codyn rawc comparing different branching factors (1 and 5) and the effect of sparsity optimization. bf 1 and bf 5 are respectively branching factors 1 and 5, and sp/nsp indicates respectively sparsity optimized and not sparsity optimized. The first case (in blue) is identical to the rawc case shown in figure 2.17. As shown, both the branching factor and the sparsity optimization have a large impact on general performance. 82

2.19 Screenshot of the cȯdyn graphical user interface. The canvas represents the cȯdyn network and can be interacted with to create and modify nodes and edges. Variables can be added, removed and inspected in the bottom panel. 83

2.20 Screenshot of the cȯdyn online playground. The panel on the left shows the cȯdyn declarative language. A rendering of the structure of the current network is shown in the bottom right panel. Once simulated, resulting signals are automatically plotted in the top right panel for inspection. The cȯdyn network can be downloaded or easily shared online by obtaining a permalink to the playground document. 84

3.1 Representation of a solution program generated by genetic programming for the canonical problem of mathematical function fitting. Mathematical operators, variables and numerical constants make up the alphabet of the genetic program. The resulting mathematical expression represented by the program can be evaluated to obtain the quality of the solution. 92

3.2 PSO optimization of the Six-hump camel function as defined in equation 3.5. This function has two global optima, at approximately the top and bottom center of the space shown here. The particles start out with random initial positions and velocities in the 2D parameter space. Particles then start to explore the parameter space based on their local and global best known parameters. As the iterations progress (left to right, top to bottom), particles start to converge on one of the global minima of the function. 94

3.3 Example parameter configuration of a single particle. Each of the parameter *pools* **A**, **B** and **C** depict a discrete number of parameter *groups*. The group number is indicated in the superscript of each box as well as by the background shading for clarity. In each pool, only one group can be active and optimized at a given time. Parameters can overlap between different groups as can be seen in pool **B**, where a valid set of parameters is either (4, 5), (5, 6) or (6, 7). One complete *subspace* is composed of selecting one group for each pool, for example {(1), (4, 5), (9)}. There are a total number of 9 parameters in this example. 97

3.4 Schematic overview of the two-layered algorithm. Each of the subspaces contains an independent PSO with a population set to the particles which are currently in the subspace. The green (triangle) particle represents the best known solution for each subspace which we call X_s (this is equivalent to X_g in the base PSO). The blue (rectangle) particle represents the globally best known solution taken over all the subspaces and is only known only to the outer layer algorithm. We call this solution X_g. 99

3.5 An example of the migration probabilities involved in migrating the pool **B** $\in s$ from one particular current group (B_c) to each possible group of **B**. The probabilities $P(B_1 \to B_1)$, $P(B_1 \to B_2)$ and $P(B_1 \to B_3)$ can be calculated using equation 3.6. The resulting probabilities (as functions of P_e, P_l and P_g) are given in equation 3.7. 101

3.6 Mutation probability characteristics for the exploration probability P_e, local exploitation probability P_l and global exploitation probability P_g, emphasizing early exploration and late convergence. 103

3.7 Particle flow for subspace 1 (left) and subspace 2 (right). The green (upper) and orange (lower) areas show respectively the in- and out-flow of particles in each subspace. 104

3.8 Comparison of the effect of two projections of multiple objectives. On the left, a weighted sum aggregation leads to a linear trade-off surface between the two objectives. While on the right, a weighted product creates a non-linear trade-off surface. Lines on both plots indicate equivalent objective values while arrows indicate the gradient of the objective function. 108

3.9 Separation of the Six-hump camel function into contributions from (left to right) x, y and xy. 110

3.10 PSO optimization with lexicographic ordering on the Six-hump camel function as defined in equations 3.14 to 3.17. The particles start out with random initial positions and velocities in the 2D parameter space. Particles then optimize for f_x, f_y and f in sequence and finally converge on one of the global minima. . . 111

3.11 Schematic overview of the optimization framework architecture 113

4.1 Kinematic and inertial properties of the biped model. Kinematic quantities are proportional to the total model height and the mass is given proportional to the total model mass. r_g indicates the segment radius of gyration, from which the inertia can be derived. The schematic on the right shows approximate center of mass locations and contact point locations on the foot (triangles). 123

4.2 Nominal joint angle trajectories, taken from Winter (2009). The dashed lines show the result of piecewise interpolating the data points, resulting in a sufficient representation of the nominal joint angle trajectories. 125

4.3 Example hip stiffness signals resulting from the *const* (blue solid), *step* (green dashed) and *ppoly* (red dotted) control modes. The large dots represent the control points which need to be optimized to obtain the resulting signals. We only annotated one pair of parameters per signal in the figure for clarity (e.g. t_2 and k_h^2 are not shown). Note that we need $N+1$ parameters for δx and N for y (here shown for $N = 4$) for the *ppoly* signal. 128

4.4 Snapshots of one gait cycle of the best obtained solutions from optimization for each control mode. The gait is shown from heel-strike to heel-strike. All three gaits show similar characteristics and look qualitatively human like. The *ppoly* gait shows full knee extension, while the other two control modes show the knee slightly flexed. The torso leans slightly forward in all cases. 129

4.5 Obtained joint angle kinematics after optimization. All angles are shown in degrees. The dotted lines are average human joint kinematics for normal walking (Winter, 2009). The solid lines are resulting joint kinematics after optimization, for the 5 best obtained solutions. Each row shows one of the *const*, *step* and *ppoly* control modes. The hip kinematics strongly correspond to normal human walking. For the knee, best results in terms of matching kinematics are obtained for the *ppoly* control mode. Ankle kinematics on the other hand are more consistent for the *step* control mode. 130

4.6 Optimized joint control signals. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$. The *const* control mode stiffness is high for both the knee and hip joint, resulting in a close match between reference and actual joint angles. The *step* control mode features relatively low stiffness patterns, while still providing close tracking of the reference joint angle. Both the hip and knee joint tend to stiffen during heel-strike, presumably to stabilize ground impact. 131

4.7 Developed by the Istituto Italiano di Tecnologia, the CoMan is a child size, humanoid robot with series elastic compliant actuators. It has 23 degrees of freedom, among which there are 6 in each leg. a) an earlier version of the robot without its covers (image taken from the IIT website). b) the CoMan version at EPFL. 135

4.8 Kinematic and inertial properties of the CoMan model. Kinematic quantities are proportional to the total model height and the mass is given proportional to the total model mass. r_g indicates the segment radius of gyration, from which the inertia can be derived. The schematic on the right shows approximate center of mass locations and contact point locations on the foot (triangles). 136

4.9 Snapshots of one gait cycle of the best obtained solutions from optimization for each control mode. The gait is shown from heel-strike to heel-strike. Note that gaits are normalized to show a single step, their respective walking speeds are $0.48\,\mathrm{m\,s^{-1}}$, $0.44\,\mathrm{m\,s^{-1}}$ and $0.33\,\mathrm{m\,s^{-1}}$. 139

4.10 Optimized joint control signals. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$. 140

4.11 Resulting control torques for the best solutions obtained for the three control modes. Torques are within limits in most cases, except for the hip torque in the *ppoly* mode and the knee in the *const* mode. The *step* torques are within range of the limits for all joints. Large torques are consistently required during mid to end stance for the ankle. 141

4.12 Fitness progression while optimizing for the last lexicographic objective, cost of transport, for the run with the best obtained fitness for each control mode. From top to bottom, the progression of respectively *const*, *step* and *ppoly* is shown.144

4.13 Optimized joint control signals of the best solutions for the *const* and *step* control modes during the last two periods of locomotion. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$. The magnitude of the perturbation force during walking, as applied to the ankle, is shown in the bottom plot. The shaded areas indicate the swing phase. 146

5.1 Existing exoskeletons for human augmentation. From left to right a) the BLEEX exoskeleton (source http://bleex.me.berkeley.edu/), b) the Sarcos exoskeleton (source http://robohub.org), c) the MIT exoskeleton (source Walsh *et al.* (2007)) and d) the HAL exoskeleton (source http://www.cyberdyne.jp/) 150

5.2 Lower extremities orthosis used for rehabilitation and support of paraplegics. From left to right a) the Ekso by Ekso Bionics (source http://www.eksobionics.com), b) the ReWalk from Argo Medical Technologies (source http://rewalk.us, courtesy of Argo Medical Technologies) and c) the Rex from Rex Bionics (source http://www.rexbionics.com) 150

5.3 Schematic representation of the attachment joints at which the wearable robot can be attached to the human body. 155

5.4 Schematic representation the realization of a specific *morphology* (right) from a kinematic *topology* (left). On the right, the *topology* is represented as a graph where edges represent joints and vertices represent segments. One possible realization of the *topology* is shown on the right. Joints in orange are the loop closure joints. 156

5.5 Morphological representations of enumerated topologies having 3 wearable robot joints and 4 wearable robot segments. Only a single realization for each topology is shown, and it should be noted that different realizations of the same topology can lead to wearable robots with significantly different properties. Joints in green (1, 4, 5 and 6 in the top row, and 3, 4, 5 and 6 in the bottom row) are the added attachment and wearable robot joints, while the orange joints (2 and 3 in the top row, 1 and 2 in the bottom row) are the added loop closure joints.158

5.6 Schematic illustration of the determination of valid actuator pairs in given a topology. In this case, the pairs $(1,4)$ and $(1,6)$ are checked for topology 2. We start on the left with the full topology and then collapse joint 1 to obtain the second topology and check for the mobility of the topology using equation 5.1. If the mobility has reduced by a single degree when collapsing a joint, then that joint can be a valid actuator. Next we check for the second actuator in each pair. When collapsing joint 4 (top right), we obtain $k = 1$ and thus DOFs are not reduced and the pair is invalid. On the other hand, collapsing joint 6 (bottom right) results in $k = 0$ and thus $(1,6)$ is a valid actuator pair. 159

5.7 Occurrence of a simple kinematic singularity. It is intuitive in this case that applying a torque on joint 6 would result in only linear forces acting on the knee joint, and not in any torque. Therefore, no amount of torque applied to joint 6 will result in any movement of the knee, hence the singularity. Note that a singularity is not a property of the kinematic chain only, it depends on where torque is being applied in the chain. In this case, the knee joint itself (for example) can still be actuated without problems. 161
5.8 Probability curves determining the rate at which particles are mutated to a different parameter subspace. The curves are designed such that there is early exploration and late convergence towards a global optimum. Since the optimization problem is complex, particles mutate on average only every 20 iterations, as determined by P_e in the beginning. From iteration 500, particles start to transfer to the globally best known parameter subspace by P_g. 172
5.9 Morphology of the 3 best obtained results. From left to right, topology 2, topology 5 and topology 3. Note that the wearable robot segments are offset in this schematic view from their original position, for clarity. 172
5.10 Number of evaluations in each parameter subspace. Each topology (on the x axis) corresponds to the topologies shown in figure 5.5, while each actuator pair corresponds to the pairs listed in 5.1. Cells indicated with 0 are invalid configuration spaces (i.e. topology 1 has only 9 possible actuator pairs). Furthermore, cells indicated with x are not visited at all while the cell marked with † indicates the topology and actuator pair of the optimal solution found in that particular run.174

B.1 Rendering of a walking sequence optimized for the CoMan robot using a variable stiffness and damping controller (*step*). The gait shown here is relatively slow, $0.44\,\mathrm{m\,s^{-1}}$, but manages an efficient cost of transport. See also section 4.2 for more details on the methods used to optimize this gait. 186
B.2 Rendering of a walking sequence optimized for the CoMan robot using a variable stiffness and damping controller (*step*) while under perturbation. The perturbations are applied randomly on the ankle in the direction of locomotion during the swing phase. The resulting controller, although open loop, is able to self-stabilize by modulating the stiffness and damping periodically. See also section 4.2.4 for more details on the methods used to optimize this gait. 187
B.3 Rendering of a non-anthropomorphic wearable robot co-designed for human locomotion. See chapter 5 for more details on the methods used to optimize this wearable robot. 188
B.4 Rendering of a non-anthropomorphic wearable robot co-designed for human locomotion. See chapter 5 for more details on the methods used to optimize this wearable robot. 189

List of Tables

2.1 List of built-in operators 16
2.2 List of built-in functions 16
2.3 Matrix index operations 19
2.4 Symbolic operations 20
2.5 Integration procedure 23
2.6 Supported generator syntaxes 30
2.7 List of common selectors 35
2.8 Commonly used spatial operations 54

3.1 Multi objective PSO methods 109
3.2 Lexicographic objectives of separated six-hump camel function 110
3.3 Task Message 113
3.4 Result Message 113

4.1 Control modes for k (same for b) 125
4.2 Lexicographic Objectives 127
4.3 Optimizing Parameters 127
4.4 Average Performance 130
4.5 Lexicographic Objectives 138
4.6 Gait speed characteristics of best obtained solutions 140
4.7 Optimization summary 144

5.1 Actuator pairs for each topology 160
5.2 List of open parameters 166
5.3 Lexicographic Objectives 168
5.4 Top 3 obtained power solutions 171

Listings

2.1 Accelerating point mass [play] 26
2.2 Two accelerating point masses with friction [play] 27
2.3 Coupling by a spring between two point masses [play] 29
2.4 Basic generator syntax [play] 30
2.5 Generating many nodes and edges using generators [play] 31
2.6 Use of expansion contexts [play] 33
2.9 Specify the type of integrator [play] 33
2.7 Basic selector syntax [play] 34
2.8 Templates [play] 34
2.10 Basic usage of events [play] 36
2.11 Basic network of coupled Hopf oscillators [play] 39
2.12 Van der Pol oscillator templates [play] 39
2.13 Matsuoka oscillator template [play] 41
2.14 Example model of a morphed phase oscillator [play] 42
2.15 Spring loaded inverted pendulum [play] 44
2.16 General physical body and joint definition 56
2.17 Various available joint types [play] 58
2.18 Simple multi-pendulum model definition [play] 60
2.19 Example usage of Jacobians [play] 65
2.20 Example of a loop closing joint [play] 69
2.21 Example of a closed loop pantographic leg [play] 87
A.1 Basic CoMan model definition in cȯdyn. This makes use of an additional file (provided below in model A.2) which specifies the inertial and kinematic properties. This file constructs the rigid body dynamics on top of the base model. 181
A.2 CoMan inertial and kinematic properties. This file is used by model A.1 to construct the full rigid body dynamics model. 182

1 Introduction

"If we knew what it was we were doing, it would not be called research, would it?" — Albert Einstein. Taken with a little smile, I personally like this expression of what it means to do research or science in general. It is of course not that we have absolutely no idea what we are doing, but it is true that we often do not know exactly where it will lead us. Exploration and discovery, challenging the status quo is at the very heart of the motivations of so many people in the world contributing to the sciences.

Scientific contributions are made in many different ways, but what they have in common is that they seek to further human knowledge and understanding of the universe. Science is in the service of society and... My apologies, I believe I let my thoughts drift there for a moment. Just like the work presented in this thesis, many scientific contributions, apart from the occasional real geniuses, are not of mind-boggling, society altering nature. Still, I want to believe that, even if ever so small, contributions can be made that serve the betterment of society in general and exploring new avenues in the process.

My own journey, written here to be read, began with an interest in the application of computer science to the human sciences, with a particular interest in human locomotion. Having a background in Artificial Intelligence rooted in social sciences rather than the usual computer sciences, I am especially interested in computer aided design methodologies based on natural processes, and apply those to the design of wearable devices for the support of the less able.

1.1 Co-design of human assistive devices

Towards this end, and as part of the European funded EVRYON project, we start by looking at a novel design methodology for the development of a lower extremities wearable robot designed for the support of human walking. To the best of our knowledge, all recently developed exoskeletons, whether for human performance augmentation or assistance/rehabilitation purposes, follow an anthropomorphic design, i.e. following the morphology of human limbs. This is certainly the more intuitive choice, since 1) the mechanical design is already known beforehand (although of course also not trivial to realize), and 2) an anthropomorphic device is more socially acceptable. On the other hand, non-anthropomorphic designs have possible advantages as well. For example, by allowing additional kinematic freedom, it is possible to increase user comfort (Schiele and van der Helm, 2006) by avoiding macro- or micro-misalignments with human joints, which behave more complexly than a single degree of freedom. Allowing more freedom in the mechanical design by avoiding anthropomorphicity also has the potential of a device which has improved or more finely tuned dynamic properties when considering the human body and the wearable robot together. We will discuss existing wearable devices for locomotion assistance in more detail in chapter 5.

Although the possibility of a non-anthropomorphic design is an interesting one, it is also more difficult to have a good intuition for what such a design should look like. In the early work of Sims (1994b,a) it was shown that by using artificial evolutionary processes, the morphology of (relatively simple) creatures could be evolved to accomplish various tasks, such as walking, jumping and swimming. The resulting morphologies gave insight into the role of the body in interaction with the environment aiding to solve these tasks. These insights may be intuitive in hindsight, but they are not necessarily so beforehand. Other seminal work in the exploration of morphology is presented in (Lipson and Pollack, 2000) where robotic lifeforms were not only designed in an automated fashion, but also manufactured. In Paul and Bongard (2001) it was shown that similar principles could aid in the design of a biped walker where morphology was adapted to show an increase in performance.

The idea that the body, and not just the control, is an important property of a system is captured by the concept of *Embodied Intelligence*. Rooted in philosophy, but applied to robotics and artificial life forms in general (Brooks, 1992; Pfeifer *et al.*, 2007) it states that artificial intelligence can exist only through an embodied agent interacting with its environment. I.e. part of what we consider intelligent behavior is caused by the body, rather than the mind (or control). It is easy to see how in nature, the body plays a large role. It is after all the only instrument through which it is possible to interact with the environment. Not only does this idea transcend to the design of robots (Hara and Pfeifer, 2003), it also becomes relevant when looking at adaptation of behavior due to unexpected changes in morphology as shown in Bongard *et al.* (2006).

Evolutionary algorithms provide an interesting approach to the goal of embodied design since they allow for very open-ended specifications of the problem domain, therefore promoting the possibility of discovery of novel designs (Floreano *et al.*, 2004). Here we seek to use these type of evolutionary processes for the co-design of a non-anthropomorphic wearable robot designed for the task of human locomotion assistance. We therefore do not only try to optimize the control of such a device, but its morphology (i.e. mechanical structure) and actuator placement as well. We look specifically at the development of a methodology grounded in an iterative design principle. Rarely are evolutionary algorithms used for the development of a final and completely finished product, nor do they allow for incorporation of all possible design parameters. Instead, found solutions give insights in possible designs which have to be further refined afterwards. To this end, we look at developing a methodology which allows for easily (re)optimizing such refinements iteratively.

1.2 Human locomotion

The co-design of a wearable device for human locomotion assistance cannot be done without looking first at the dynamics of human locomotion itself. Human locomotion is a well studied subject and of particular interest is the early work done on passive dynamic walkers (McGeer, 1990). Here it is shown that much of the dynamics of walking can be prescribed to the mechanics, instead of actuation or control. This idea is very much in line with the idea of embodied intelligence since it turns out that the human body, although not entirely passive of course, is particularly well suited for the task of locomotion. Passive dynamic walkers are however also very unstable (at least the ones that are purely passive). The slightest disturbance could destabilize the system and make the walker fall over easily. A wearable device in many ways can be seen as a disturbance to the human body and it would therefore make sense to try and optimize not only the control of a wearable device but also its structure such as to find new optimal locomotion modes, such that disturbances are minimized.

Since open-ended optimization is a complex task, especially when looking at optimizing for stable bipedal gaits, we first seek to verify our methodology on normal, un-augmented human walking. We are interested in the minimal conditions for the emergence of human gait by optimizing high-level objectives only, and look at the application of population-based optimization strategies towards this goal. If we manage to do so, then within reasonable assumptions, we can try and apply the same methodology to the co-design of a wearable robot. To validate the importance of morphological design for the emergence of natural gait we also look at applying the exact same methodology to a model of a humanoid robotic platform and see that it is important to not only mimic natural systems, but take dynamical properties of such a system into account during its design.

1.3 Tools

In the process of developing the necessary tools for doing the research as described in the previous paragraphs, we realized that the available scientific tools needed for our approach were not readily available. We firmly believe that one of the core values of scientific work lies in the fact that research should be open, freely available to everyone and anyone, and readily reproducible. We realized after our initial studies that this would not be easily possible with available simulation software. Furthermore, we found that most existing software was not adequate for the use of morphological design of our specific needs.

In particular, we look at advancing the state of the art in two areas of interest, by providing ready to be used, open, and freely distributed frameworks specifically designed for the research presented in this thesis. The first area of interest is that of general purpose, unified coupled dynamical systems modeling, including rigid body dynamics. Specifically when looking at the simulation of 1) complex rigid body systems, such as parallel structures, 2) contact models suitable for locomotion and 3) simulations targeting design principles, we found none of the existing solutions suitable for all of our criteria. We are not the only ones that have shown a recent interest in the scientific community for the availability of such simulation software and we believe that with the work presented in this thesis we make a valuable contribution. To the best of our knowledge, we present here the first freely available and open implementation of a novel dynamical system simulation environment with a strong focus on openness, expressiveness, modeling, performance and education. We focus specifically on the modeling and simulation of coupled dynamics, whether coupled oscillator systems (or central pattern generators) , rigid body dynamics or other types of dynamics.

The modeling of rigid body dynamics is certainly not a new problem. There are generally speaking two methods for deriving the dynamical equations that govern rigid body systems. Simulators such as ODE, Bullet or Box2D are open and freely available simulators which initially consider all physical bodies to be unconstrained and then explicitly add constraints to the individual body equations of motion. This method is popular in these engines, which target games and movie production, since it leads to physically believable and relatively fast forward simulations. They are however inadequate when used for research purposes, in particular when looking at design. The reason is that all of these simulators aim at physically realistic simulations, but not necessarily accurate ones, often trading accuracy for speed of simulation. They also do not derive equations of motion in a form useful for model-based control, deriving interaction forces, accurate contact modeling or system analysis.

On the other end of the spectrum are simulators which actually derive the system's equations of motion, whether symbolically or numerically, in its entirety by projecting the dynamics into generalized coordinates. This has many advantages, in particular for research purposes since analysis can be done on the system when all quantities of the equations of motion are known. Furthermore, the resulting dynamics are physically more accurate and useful quantities for design of systems. For example accurate interaction forces are readily available. The derivation of the equations of motion is a well studied subject and an outstanding and detailed explanation of it can be found in Featherstone (2008). Few implementations however are available at present. Of those, including simulators such as Robotran or OpenSim, none provide 1) ease of modeling, 2) support for sophisticated contact modeling, 3) closed loop dynamics, 4) performance and 5) easy extensibility. Additionally, recent developments such as MuJoCo (Todorov *et al.*, 2012), which take a similar approach, are of great interest but unfortunately as of yet are unavailable. Not only do we aim at providing a state of the art implementation of the derivation of the equations of motion of rigid body systems as detailed in Featherstone (2008), we importantly do so in a manner which makes modeling pleasant and unified with other types of dynamical systems.

The second area of interest is that of large scale, population-based optimization. Although not necessarily novel, we develop a framework which allows for the managed execution of multi-user, large scale population based optimizations. Furthermore, the framework is agnostic in terms of optimization algorithm or task execution and can be used for general purpose task distribution. Several optimization algorithms including basic Genetic Algorithms and Particle Swarm Optimization are provided, as well as several common task dispatchers such as simulation in the Webots (Michel, 2004) simulator or Matlab. We understand that the scientific value of such a framework is limited, however it has been of integral importance as an application to many of the research performed in our laboratory and in particular to the work presented in this thesis. We also believe that as such it has potentially great value for the scientific community.

1.4 Organization of the thesis

The remainder of this thesis is divided into two parts. In the first part we develop tools, frameworks and algorithms for the modeling and simulation of coupled dynamical systems and look at the use of population based, large scale optimization specifically for use in robotics.

In chapter 2 we first contribute a state of the art, open and freely available framework for the modeling of coupled dynamical systems, with a specific focus on central pattern generators and rigid body dynamics. We develop a novel declarative modeling language in which coupled dynamics can be naturally and uniformly described, allowing integration of multi-domain dynamical systems. Using this language, a modern and competitive rigid body dynamics simulator is implemented based on Featherstone (2008), including declarative system modeling, custom joint models, soft and hard contact models, inverse- and forward-dynamics and modeling of closed loop systems. Finally, optimized, low-level code is automatically generated from the high-level model description with a special focus on running on Real-Time and/or embedded systems, even micro-controllers.

We then continue to chapter 3, where we briefly discuss population based optimization methods and develop a novel Particle Swarm Optimization (Kennedy and Eberhart, 1995) based algorithm for the simultaneous optimization of solution structure and its corresponding continuous parameters. This allows for the optimization of the type of problems which have a *known* set of possible solution structures each with a set of, possibly overlapping, continuous parameters. Finally, we present another open and free to use framework for performing large scale optimizations in multi-user environments, particularly suited for population based optimizations.

In the second part of this thesis we use these tools to study the co-design of a non-anthropomorphic, lower extremities, wearable robot for human locomotion assistance. Because natural human gait lies at the core of human locomotion assistance, before we dive into the co-design, we first explore in chapter 4 the use of population based optimization methods and impedance control to (re)discover natural human gait from first principle objectives only. We show that global gait characteristics, such as heel-strike, toe-off and stance/swing duration, and stable walking are obtained by optimization for forward locomotion and energy efficiency only, and look at the role of impedance control to do so. The same method is then used to optimize a human like gait (in simulation) for the CoMan compliant humanoid robot (Tsagarakis *et al.*, 2013) by looking a the optimization of mechanical cost of transport. Here we show by comparison with our previous study on a normal human sized model, that the morphology (and not only the biomimeticity) of a humanoid robot plays an important role in its design. Finally, the role of impedance control in locomotion while under the influence of perturbations is studied using these same methods.

In chapter 5, we investigate the use of population based optimization algorithms for the co-design of a wearable robot suitable for human locomotion assistance. We look at optimization of the morphology of the human augmenting robot to assist locomotion using a semi open-ended explorative search with the algorithm developed in chapter 3. Together with the tools developed in chapter 2 we provide a comprehensive framework for the future exploration of iterative design methods for non-anthropomorphic wearable robots.

The contributions made in the course of this thesis are the following:

1. A state of the art framework for the modeling and simulation of coupled dynamical systems using a novel design methodology for model construction allowing for the construction of complex and parametrized systems. Using a unified approach for the modeling of dynamics, multiple dynamical domains can be modeled in the same way. In particular, we focus on the modeling of 1) oscillators and central pattern generators and 2) rigid body dynamics. Special attention is paid to the simulation of dynamical systems suitable for Real Time and embedded systems as well as micro-controllers, without the loss of expressiveness or generality.
2. A novel Particle Swarm Optimization based optimization algorithm for the simultaneous optimization of structural parameter configuration spaces and their corresponding (possibly overlapping) continuous parameters. The resulting algorithm is suitable for optimization problems in which the possible set of solution structures is known (or can be enumerated) but is to be explored using cooperative strategies similar to those of a normal Particle Swarm Optimization.
3. A framework for performing task and task execution agnostic, large-scale, population-based optimization in a multi-user environment. Combined with the framework for modeling of coupled dynamics, the two frameworks provide a methodology for the optimization of various robotics problems, of which we present two in particular.
4. A method for the optimization of natural human gait using impedance control from high-level objectives (such as uprightness, walking at a desired speed and minimization of energy). The proposed method leads to the automatic recovery of global human gait characteristics, such as swing/stance duration and heel-strike, foot-roll and toe-off without explicitly optimizing for it. Furthermore, using the same methodology, we show the importance of morphological design by optimizing for natural gait of a humanoid robotic platform, and explore the role of impedance control further under the

application of perturbations and rejection of its disturbances.

5. Finally, we combine all of the above developed methods for the design of an iterative methodology and employ the resulting system for the co-design of the morphology and control of a wearable robot for the purpose of human locomotion assistance. We show the viability of such a methodology and provide insight in possible non-anthropomorphic design principles.

We conclude the work presented in chapter 6.

Dynamics and optimization Part I

2 Dynamics

Dynamics are everywhere, and quite literally so! Merriam-Webster defines *dynamics* as “a pattern or process of change, growth, or activity”. In a sense, a dynamical system can be defined as a system that is subject to change over time. Obviously, this is a very general definition that is generally true for most systems (at least the interesting ones). From a mathematical point of view, a dynamical system is simply any fixed set of rules which describe the time dependence of the position of a point in a space.

This chapter surveys the simulation of a particular set of dynamical systems, namely those systems described by Ordinary Differential Equations (or ODE’s). When we originally set out, we found that existing software for working with these type of systems, although providing great tools for some of its aspects, would on the other hand sorely lack other important features, such as ease of modeling, good performance, availability and model parametrization. Of course, all major scientific computing software packages such as Matlab or Mathematica allow you to write down the differential equations that govern a particular system and (for example) numerically integrate them. This however is not where great software assisted design of dynamical systems should end.

A well designed software framework would allow you to write differential equations in their pure mathematical form and couple isolated systems together as a first class construct. It would allow you to define your system in a structural and maintainable way while visualizing, analyzing and numerically integrating them. It would then allow you to take your high-level, structurally and mathematically sound system and automatically transform it into an efficient representation suitable for running on constrained embedded and/or real time hardware (with a particular focus on robotics systems). It would also provide a multitude of tools for graphically designing simple systems (great for educational purposes), and bind to various existing programming languages with minimal effort.

Dramatism aside, ċodyn is a software framework that features all of the above, and more. In many ways, it has been the cornerstone of this thesis and deserves to be its first chapter after the introduction. In the remainder of this chapter we go through its major design aspects, various implementation details and give a number of examples of systems which are ideally suited for representation and simulation in ċodγn.

2.1 ċodγn, coupled dynamics

ċodγn (**Co**upled **Dyn**amics) is the umbrella name for the framework of various software libraries and tools specifically designed to address issues with existing frameworks to design and simulate coupled dynamical systems. The core motivations that drove the design for such a framework are the following:

1. *Free/Open*: The most popular available scientific tools which do well at dynamical system modeling are proprietary. Open alternatives such as Octave, or more recently Julia are available but lack libraries, toolboxes and communities to make them real alternatives in many cases. Scientific advancement is an open endeavor at its very core and its tools should therefore also be available in the open, free (as in freedom) for anyone to inspect and modify. Even if personally a strong proponent of free/open software, proprietary software is not only problematic from an ideological standpoint. There has been a recent push towards explicitly producing free and open software as a product of scientific projects (as a European policy) for a good reason. Open software can be reused, modified, improved and scientific work done with them easily reproduced without requiring an expensive license. Although there are free alternatives for tools such as Matlab and Mathematica (for example Octave or Maxima) they are not specifically tailored towards simulations of dynamical systems and do not meet our other motivations.
2. *Domain specific*: Domain specific languages (or DSLs) are languages specifically designed to fit a particular domain. They are popular because they can be made such that their problem domain can be represented in the best suitable way. This is in contrast to general purpose languages which have to cover general computational requirements.
3. *Expressive*: Users should be able to express a dynamical systems model in a concise manner, without losing flexibility in general. Existing software frameworks do not provide specific constructs to express complex dynamical systems in a short and easily understood way.
4. *Performance*: There is often a trade off to be made between 1) expressiveness/flexibility, 2) ease of use and 3) performance. It is relatively easy to create a system which features two of these characteristics, but when a framework is focused on the first two characteristics performance is often lacking. ċodγn aims to provide good performance while preserving expressiveness and ease of use.
5. *Educational*: ċodγn was not only created as a research tool but also as a tool for educational purposes. It provides various utilities which make it easy to explore and interact with dynamical systems. Although ċodγn has not been used as course material, we have used it to introduce modeling and simulation of coupled dynamical systems for semester and master projects where it provided a great learning experience through

experimentation.

The next sections will first go over the conceptual design principles of modeling in cȯdүn and the specially developed language which implements this design. Then two realizations of coupled dynamical systems, central pattern generators and rigid body dynamics are discussed within the developed framework. Finally details on performance, tools and examples are provided.

cȯdүn is available at http://www.codyn.net/. The website contains all information related to cȯdүn, including documentation, manuals, examples, downloads and sources. All of the cȯdүn software is released under the LGPL (for libcodyn) and GPL free software licenses.

2.2 Core concepts

cȯdүn is specifically designed to ease the modeling of coupled dynamical systems. We focus only on systems which can be represented by sets of ordinary differential equations. Although partial differential equations underlie many naturally occurring dynamical systems as well, cȯdүn currently does not support their modeling. Coupled dynamical systems are systems which can usually be designed as a number of independent, isolated systems plus their intercoupling. Often, these couplings can be represented as additive terms in the differential equations of the separated systems. This leads to a natural representation of these systems as a layered directed graph, which is one of the core foundations of the cȯdүn framework. It should be noted that cȯdүn only supports *additive* coupling naturally. Other types of coupling, such as multiplicative coupling involving several emitters, can still be used, but not without working against cȯdүn's main concepts.

This section briefly introduces the core concepts and principles upon which the cȯdүn framework is built.

2.2.1 Nodes and edges

cȯdүn has been designed around concepts which give an intuitive notion to building coupled dynamics system. In cȯdүn, a coupled dynamical system is called a *network*. The *network* term arises from the fact that in cȯdүn, dynamical systems are modeled using *nodes* and *edges*, which are structurally organized as a network (or graph) of connected components.

A node in cȯdүn is simply an object containing variables. There are several types of variables with different semantics. A **state** variable is a variable with an associated differential equation which can be numerically integrated over time. A **discrete** variable is similar to a **state** variable, except that it describes the discrete-time system (map) differential equations instead of a continuous integral. Finally, there are **normal** variables which can be used to define reusable (sub)expressions or parameters.

An edge connects two nodes and defines a coupling of variables between nodes. For **state** variables, an edge defines a (part of a) differential equation of the state variable. Edges can reference variables from both input and output nodes and a single edge can define differential equations for more than one variable.

It is important (conceptually) to realize that all differential equations in cȯdγn are implemented using edges. To make writing differential equations which only access states and variables from the same node more convenient, each node also contains a so-called self-edge. The self-edge is basically an edge with its input and output set to the node it is contained within.

To allow composability and modularity, nodes can themselves contain other nodes and edges. By these means, subsystems can be easily constructed and interconnected. The network itself, then, simply becomes the top level node containing top level variables and other child nodes and edges.

Consider for example the following simple system of coupled differential equations:

$$\dot{x}_1 = -1 + (x_2 - x_1) \tag{2.1}$$

$$\dot{x}_2 = 1 + (x_1 - x_2), \tag{2.2}$$

with initial conditions $x_1 = 1, x_2 = 0$. This system can be represented naturally in cȯdγn by separating the coupling terms from the "main" differential equations for each variable:

$$\dot{x}_1 = -1 \tag{2.3}$$

$$\dot{x}_2 = 1 \tag{2.4}$$

$$\dot{x}_{2\to 1} = x_2 - x_1 \tag{2.5}$$

$$\dot{x}_{1\to 2} = x_1 - x_2 \tag{2.6}$$

Of course, this is a somewhat contrived example and coupling could be separated differently or not at all in this case. Equations do not need to be separated in this way, and the user is free in which manner the equations are modeled conceptually. However, as we will see later, the separation of equations in their canonical system dynamics and coupling dynamics is natural for a variety of systems. We will see more examples of coupled dynamical systems in the following sections. Conceptually, this system can be modeled in cȯdγn using nodes and edges as shown in figure 2.1.

cȯdγn requires the user to specify systems in this manner, i.e. using nodes and edges, and a large number of systems can be modeled naturally using these concepts.

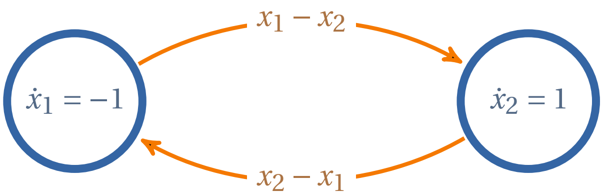


Figure 2.1 – A conceptual representation of a network of two nodes which are bidirectionally coupled.

2.2.2 Mathematical language

All variables and differential equations are expressed in a mathematical expression language which closely resembles that of existing programming languages, with some additional features specific to the design of dynamical systems. In cȯdyn everything that has to be computed is expressed by mathematical expressions. In addition, all values in cȯdyn are real-valued, 2-dimensional matrices. This decision imposes certain limitations. For example, complex numbers cannot currently be represented in cȯdyn, nor can natural or integer numbers. However, we found that this does not pose a practical limitation on the type of systems that we want to model in cȯdyn. Vectors are simply a matrix of n-by-1 or 1-by-m and single numerical values are matrices of 1-by-1.

Furthermore, all mathematical expressions have static dimensions. In other words, once an expression is defined, its dimensions cannot change during the course of simulation. This is a very important design decision in cȯdyn leading to a large number of advantages in terms of performance and applicability. The rationale, advantages and disadvantages of this are discussed in more detail in section 2.7.

Built-in operators

cȯdyn supports all of the standard mathematical operations on values and all operations are properly defined for vectors and matrices. Table 2.1 lists all available operators in the language. Unless otherwise specified, they operate on an element-wise basis. For binary operators, the left and right hand sides need to be of the same dimension (with the exception of matrix multiplication) or one of the values needs to be a 1-by-1 value.

Logical operations are supported, but operate on floating point values. The resulting values are again floating point values.

Table 2.1 – List of built-in operators

Operator	Description	Matrix behavior
$-a$	Unary minus	
$a-b$	Subtraction	
$a+b$	Addition	
$a*b$	Multiplication	Matrix multiplication is performed when the number of columns in *a* equals the number of rows in *b*. Otherwise, element wise multiplication is performed.
$a.*b$	Element wise multiplication	Unambiguously performs element wise multiplication.
a/b	Division	
$a\%b$	Floating point modulo	
$a \hat{} b$	Power	
	Logical operators	
$a>b$	Larger than	
$a<b$	Smaller than	
$a>=b$	Larger than or equal to	
$a<=b$	Smaller than or equal to	
$a==b$	Equal	
$a!=b$	Not equal	
$a\|b$	Or	
$a\&\&b$	And	
$!a$	Negation	
$a?b:c$	Ternary conditional, i.e. (if *a* then *b* else *c*)	
	Unicode operators	
$a\cdot b$	Multiplication of *a* and *b*, same as ∗	
$a\div b$	Division, same as /	

Built-in functions

Table 2.2 – List of built-in functions

Function	Description
`sin`(*a*)	Sine
`cos`(*a*)	Cosine
`tan`(*a*)	Tangent
`asin`(*a*)	Arc sine
`acos`(*a*)	Arc cosine

`atan`(a)	Arc tangent
`atan2`(a, b)	Arc tangent of two variables
`sinh`(a)	Hyperbolic sine
`cosh`(a)	Hyperbolic cosine
`tanh`(a)	Hyperbolic tangent
`sqrt`(a)	Square root
`invsqrt`(a)	Inverse square root
`hypot`(a, b)	Euclidean distance between a and b
`hypot`(a)	Norm of a (i.e. $\sqrt{a^2}$)
`sqsum`(a)	Squared sum of a (i.e. $\sum_i a_i^2$)
`min`(a, b)	Element wise minimum of a and b
`min`(a)	Minimum element of a
`min`(a, b)	Element wise maximum of a and b
`min`(a)	Maximum element of a
`exp`(a)	Base-e exponent of a
`exp2`(a)	Base-2 exponent of a
`erf`(a)	Error function of a
`floor`(a)	Rounding down to nearest integer
`ceil`(a)	Rounding up to nearest integer
`round`(a)	Rounding to nearest integer
`abs`(a)	Absolute value
`pow`(a, b)	Power of a to b
`ln`(a)	Base-e logarithm of a
`log10`(a)	Base-10 logarithm of a
`lerp`(a, b, c)	Linear interpolation of b to c given $a \in (0, 1)$
`sign`(a)	Sign of a
`csign`(a, b)	Value of a with sign of b
`clip`(a, b, c)	Value of a bounded in (b, c)
`cycle`$(a, b, c$	Value of a cyclic to (b, c)
`index`(a, b)	Index a by indices in b
`lindex`(a, b, c)	Linear indexing of a by indices in b, given row dimension c
`transpose`(a)	Transpose
`inv`(a)	Inverse
`pinv`(a)	Pseudo inverse
`linsolve`(a, b)	Solve for x in $ax = b$
`qr`(a)	QR decomposition
`sum`(a)	Sum of elements in a
`product`(a)	Product of elements in a
`length`(a)	Largest dimension of a
`size`(a)	Dimension of a (returns 1-by-2 rows and columns in a)
`size`$(a, 0)$	Number of rows in a
`size`$(a, 1)$	Number of columns in a

`vcat`(a, b)	Vertically concatenate a and b
`zeros`(n, m)	Zero matrix of n-by-m
`eye`(n)	Identity matrix of n-by-n
`diag`(a)	Diagonal of a
`tril`(a)	Lower triangular matrix of a
`triu`(a)	Upper triangular matrix of a
`csum`(a)	Column wise summation
`rsum`(a)	Row wise summation
	Unicode functions
a^T	Transpose, same as `transpose`(a)
$\sum(a)$	Sum of elements in a, same as `sum`(a)
$\prod(a)$	Product of elements in a, same as `product`(a)
a^2	a squared, same as `pow`$(a, 2)$
$\sqrt{}(a)$	Square root of a, same as `sqrt`(a)

	Rigid Body Dynamics Functions
`slinsolve`(A, b, λ)	Sparse linear system solve for x in $Ax = b$ given sparsity induced by branching from λ
`sltdl`(A, λ)	$L^T DL$ decomposition of A given sparsity induced by branching from λ
`sltdldinv`(L, b)	Solve for x in $x = D^{-1}b$ given $L^T DL$ decomposition in L
`sltdldinvlinvt`(L, b, λ)	Solve for x in $x = D^{-1}L^{-1}b$ given $L^T DL$ decomposition in L and sparsity induced by branching from λ
`sltdllinvt`(L, b, λ)	Solve for x in $x = L^{-T}b$ given $L^T DL$ decomposition in L and sparsity induced by branching from λ
`sltdllinv`(L, b, λ)	Solve for x in $x = L^{-1}b$ given $L^T DL$ decomposition in L and sparsity induced by branching from λ

A large number of built-in functions are readily available for use in ċȯdẏn. Table 2.2 lists all currently available functions. Most functions listed are general purpose mathematical functions. There are however a few special purpose functions (at the end of the table) which are used to solve various aspects of the rigid body dynamics. More details about these functions are provided in section 2.6.

Random numbers

The built-in functions `rand`(a) and `rand`(a, b) deserve special mention in the context of a ċȯdẏn dynamical system. ċȯdẏn takes special care to ensure that random numbers can be used reliably and reproducibly during numerical integration. This needs special attention due the fact that expressions are evaluated as needed during simulation. Therefore, if a simulation was to be repeated we would like to avoid to obtain different results if expression were to be evaluated in a different order (which can happen due to the lazy evaluation behavior of cȯdγn). This is a real problem that earlier versions of cȯdγn did not properly address.

To solve this, cȯdγn keeps track of all calls to `rand` and transforms these to special instructions which will always return a cached version of their current random value. These cached random values are then updated at every integration step. Thus, if for example a variable is defined as $v = \texttt{rand}()$, then within the same integration step all references to v will observe the same random value. The random number generator used in the network can also be seeded externally such that results can be easily reproduced.

Referencing variables

Named variables can be referenced by name in any expressions. For expressions in nodes, variables are always resolved first in the same node, then in the parent node, etc. For edges, variables are first resolved in the edge, then in the input node of the edge, and then in the parents of the edge. Variables can also be referenced in child nodes by using a dot syntax (e.g. `child.v`).

Matrix indexing

cȯdγn supports indexing of matrices and vectors. Table 2.3 lists the various types of indexing that are supported. Unlike some popular languages, indices in cȯdγn start at 1.

Table 2.3 – Matrix index operations

Syntax	**Description**
$A[n,m]$	With n 1-by-1 and m 1-by-1, indexes row n and column m in A
$A[i]$	With i 1-by-1, linearly indexes A in column major order
$A[r,c]$	With r n-by-1 and c 1-by-m, indexes the cross section of row indices r and column indices c
$A[:,c]$	With c 1-by-m, indexes the cross sections of all rows and column indices c
$A[r_b:r_e,c]$	With r_b 1-by-1, r_e 1-by-1 and c 1-by-m, indexes the cross section of row indices r_b to r_e and column indices c
$A[B]$	With B n-by-m, linearly indexes A in column major order and returns an n-by-m matrix

User functions

Apart from variables, nodes can also contain user defined functions. These functions consist of a mathematical expression which resolves its variables in its named arguments. Once defined, user functions can be called in the same way as built-in functions are called. User functions cannot produce side-effects (i.e. they simply return the value of their expression), but they can resolve variables and other functions from the context in which they are defined.

Symbolic math

cȯdγn implements a small number of symbolic operations. In cȯdγn jargon, these are called operators, since they operate on symbolic expressions. Symbolic operators receive one or more expressions which they are free to operate on. They then return a function, implementing the symbolic operation, which can be called like any other function. Table 2.4 lists the most important operators available.

Table 2.4 – Symbolic operations

Syntax	**Description**
`dt` $[\mathit{expression}, n]()$	Calculates the n^{th} time derivative of the provided *expression*.
v', v''	Shorthand syntax to obtain the respectively first, second, etc. time derivative of a variable v (i.e. equivalent to `dt` $[v]()$, `dt` $[v, 2]()$, etc.)
`diff` $[f, n; v_1, v_2, ...](\mathit{args}...)$	Obtain the n^{th} symbolic derivative of the user function f, towards the variables (function arguments) v_1, v_2, etc. The resulting function represents the derivative of f and can be called with the same arguments as f
`pdiff` $[f, n; v](\mathit{args}...)$	Obtain the n^{th} *partial* symbolic derivative of the user function f, towards the variable (function argument) v.
$\partial[f, n; v](\mathit{args}...)$	A shorthand notation for the partial derivative operator.
`delayed`$[\mathit{expression}, \mathit{init}](\mathit{dt})$	Provides a *dt* delayed version of the provided *expression*. The *init* expression is optional and is used to initialize the delay history (which is 0 if not specified). The *init* expression can reference t which will run from $t_{begin} - dt$ to t_{begin}

Note that the symbolic derivation in cȯdγn is deeply integrated into the mathematical engine. This means that derivatives are properly propagated, can be taken on user defined functions and arbitrary expressions. Time derivatives properly take into account the special time variable t and properly use the differential equation of a state variable as the time derivative if required.

2.2.3 Edge projections

All differential equations in the system, i.e. the derivatives of all state variables, are written as mathematical expressions operating on **state** variables through edges. Conceptually it is useful to think of differential equations as the projected rate of change of a state variable. Edges in the network graph project these rates of change, which they encode, towards **state** variables in nodes in the system. More than one edge can project a differential equation on the same state variable. The resultant *true* differential equation is then simply the sum of all the individual projections.

Direct projections

Apart from projecting onto **state** variables, edges can also project onto *normal* variables. Since nodes implement the concept of data encapsulation (they cannot directly inspect data from other nodes), direct projections can be used to transfer *data* from one node to another. This has exactly the same effect as accessing variables from other nodes directly, except that the relationship between the nodes is now encoded by its edges. This makes it clear where data between nodes comes from in a structural manner. An example where this is useful is when integrating external data sources into a cȯdγn network. A special node can contain variables representing external data (such as sensor values) which can then be directly projected to all nodes requiring that information.

2.2.4 Events

Dynamical systems are not always modeled as one single continuous system. It is not uncommon for a single dynamical system to instead be modeled as several systems with different dynamics, and a switching mechanism to transition from one system to another (for example rigid body with contact dynamics). These types of discrete, or hybrid, dynamical systems are supported in cȯdγn by means of a built-in event system. Events are embedded inside nodes and allow a transition from one or more *event-states* of that node to another when a specified condition becomes *true*. Additionally, events can cause discrete changes in variables. This is used to allow implementation of changes in states over the horizon of the event (see section 2.6.10 for an example of how this is used when implementing hard contacts for Rigid Body Dynamics). Finally, differential equations can be associated to be only *active* when the corresponding *input* node of the edge is in one or more particular *event-states*, thus allowing for different dynamics during different states of the simulation.

Event refinement

Events can be activated by simply observing the condition at every simulation step, but it can sometimes be important to be more precise about the exact time at which the event condition transitioned from *false* to *true*. Having inaccurate event timing can lead to inaccuracies in the simulation (for example energy loss) since part of the dynamics are incorrectly simulated or can lead to penetration errors in contact modeling. In cȯdγn this is called event refinement which can be enabled for each event individually. To refine events, a maximum allowed error on the transition condition of an event can be specified. The logical condition expression is decomposed into a binary tree where each non-terminal node is a logical operator (i.e. $<, >, <=, >=, ||, \&\&$), and each terminal node is a mathematical expression with a non-logical root. The logical operators $<, >, <=, >=$ are transformed such that their zero-crossing (from negative to positive) indicates that the event condition activated during the current integration step. This can be done simply by replacing the logical operator with a subtraction (and reversing left and right hand sides for $<$ and $<=$). If we indicate this transformation by $\mathbb{Z}_i(a)$, with i the integration step and a the logical expression, then we can say that the event activated when $\mathbb{Z}_{i-1}(a) < 0$ and $\mathbb{Z}_i(a) >= 0$ (or $\mathbb{Z}_{i-1}(a) <= 0$ and $\mathbb{Z}_i(a) > 0$ in the case of the $<=$ and $>=$ operators). In other words, the event activates when $\mathbb{Z}$ undergoes a positive zero-crossing. From this representation, we can also directly obtain a *linearized* estimate of the time step $\Delta t_e(a)$ required to obtain an exact event condition, as given in equation 2.7.

$$\Delta t_e(a) = \frac{-\mathbb{Z}_{i-1}}{\mathbb{Z}_i(a) - \mathbb{Z}_{i-1}(a)} \Delta t \tag{2.7}$$

For the || operator we can simply check if either of its operands had a zero crossing. Similarly, for the && operator we check if 1) the left hand side has a zero crossing while the right hand side is positive, 2) the right hand side has a zero crossing while the left hand side is positive or 3) both left and right hand sides have a zero crossing. The estimated Δt_e of the operand for which a zero crossing occurred is propagated upwards in the binary tree. If both operands underwent zero crossings, then the smallest Δt_e is propagated.

2.2.5 Numerical integration

Numerical integration is one of the main uses of cȯdγn for dynamical system modeling. For this purpose, cȯdγn provides a flexible and extensible numerical integration infrastructure with different numerical integration methods. Special care has to be taken to evaluate the various features of the network, handling differential equations, events and random numbers in the right order. The general integration procedure is shown in table 2.5.

Only steps 5) and 6) are specific to the type of integrator used. This makes it easy to provide a variety of different numerical integrator schemes, which are only concerned about computing the derivatives and numerically integrating the states, without needing to know about any cȯdγn specific internal features. A basic variety of built-in integration schemes, including Euler, Runge-Kutta 4th order, Leap-Frog and Correction-Prediction are implemented and provided (Butcher, 2008). Furthermore, new integrator types can be easily implemented and loaded through the use of plug-ins. cȯdγn currently does not provide adaptive time step integrators, although they could be easily implemented. The reason is that it is often preferred to have guarantees on execution time, which adaptive time step integrators do not give. However, they do provide more accurate integration and can be better suitable for systems with specific behavior (such as stiff systems). It would therefore be interesting to provide adaptive time step integrators in the future.

Table 2.5 – Integration procedure

1) Compile all the equations in the network
2) Collect all **state** variables, **discrete** variables, edge projections, events and random number instructions
3) While $t <$ endtime
    1) Store current event condition expression values
    2) Generate a new set of random values for all `rand` instructions
    3) Store the current values of all **state** variables
    4) Update all symbolic math operators (for example, compute and store delayed history)

    5) Evaluate all active edge projections on **state** variables with the selected integrator to compute the **state** derivatives
    6) Numerically integrate all **state** variables with the selected integrator, using the computed derivatives from the previous step

    7) Evaluate all active edge projections on **discrete** variables
    8) Discretely integrate all **discrete** variables using the values obtained in the previous step

    9) Evaluate all event conditions and determine which events were activated by the current integration step. For all activated events, determine (if any) the smallest requested event refinement. If there is an event refinement then, restore the previously saved **state** (from step 3)) and continue from step 5).
    10) Execute all activated events, evaluating discrete changes to variables, updating the *event-state* of the nodes to which the events belong and updating the active set of edge projections according to the new *event-state*

2.3 Modeling language

The previous section gave a conceptual overview of the cȯdγn framework for dynamical modeling, but did not yet show how the actual modeling in cȯdγn is done. Domain specific languages, or DSLs, are special purpose computer languages developed to address problems in a specific domain. This is in contrast to general purpose programming languages which are able to solve problems in any domain by means of general, instead of specific, constructs. The advantage of a domain specific language is that it can be tailored to fit the problem domain *exactly*, leaving out constructs that are not required, thus resulting in smaller and easier to understand languages. At the same time, anything outside of the domain or its conceptualization will be harder to express than it would normally be in a general purpose language.

In earlier versions of cȯdyn, models were specified using an XML derived format. XML is very much a general purpose markup language which can be used to describe any type of hierarchical model, and is commonly used for defining models in existing modeling tools (for example URDF). In this format, every part of the model had to be explicitly written down and, due to the nature of XML, this quickly led to very large and hard to maintain models. It became clear that although the cȯdyn framework provided useful concepts and tools for dynamical systems modeling, the manner in which these models needed to be written down was lacking and hindered adoption.

The cȯdyn modeling language is a *declarative* DSL tailored towards the specific concepts described in the previous section. It allows for concise expression of complex, coupled dynamical models in an easy to learn language. Making the language *declarative*, as opposed to imperative (or functional), is a natural paradigm for modeling. I. e. models are described by their structure rather than as a series of steps. This makes them easier to reason about, analyze and manipulate. Not only is this important because it makes the modeling easier, it also allows for writing tools that can automatically analyze, transform and optimize models, which would otherwise be difficult if not impossible. Section 2.7 shows in detail how models can be automatically optimized for performance, while section 2.8 explains how various tools are built to work with cȯdyn models.

In the remainder of this section the specially developed cȯdyn modeling language is described in detail. Each feature of the language is accompanied by examples of increasing complexity to show how it can be expressed in the cȯdyn language.

2.3.1 Variables and simple differential equations

The basic building blocks of a cȯdyn model are *nodes*, *edges* and variables. An empty document represents the top level cȯdyn network, which conceptually is just like any other node, except that it does not have any parent node. To define variables in a node, the following syntax can be used:

```
# Position of the point mass
y = 1

# Gravity
g = 9.81

# Mass
m = 0.6

# External force
f = 0

# Acceleration
a = "-g + f / m"
```

Note that the definition of a variable consists of a name, followed by `=`, followed by the expression for that variable. Simple numerical values can be specified directly (`9.81`), but more complex expressions must be enclosed in double quotes `"like so"`.

This network in itself is not that interesting, since it does not specify any dynamics of the system. The acceleration of the point mass is defined, but just as a normal variable. To define a differential equation instead, we can use the following syntax:

```
# Initial position of the point mass
y = 10

# Gravity
g = 9.81

# Mass
m = 0.6

# External force
f = 0

# Acceleration
ÿ = "-g + f / m"

# Or alternatively
# y’’ = "-g + f / m"
```

The simple statement in the example above has several important and interesting implications. First, cȯdyn is a Unicode aware language. This means that we can use Unicode characters in names, so writing `τ = 2` is perfectly valid in cȯdyn. Furthermore, cȯdyn has Unicode syntax support for a few operations allowing to write models closer to their mathematical form. To write differential equations, cȯdyn supports the Unicode combining dot and double dot characters, i.e. one can write $\dot{y}$ to define a first order differential equation, or $\ddot{y}$ for a second order equation. It is not always convenient to write equations in this way, so cȯdyn also supports a alternative prime (`y’’`) syntax for the same purpose.

The second important implication is that although cȯdyn internally only supports first order differential equations, we can easily still easily write nth order models. cȯdyn automatically transforms the model in a series of equivalent first order equations. If we inspect the generated internal model, we can see that it has been transformed to the following equivalent form:

cȯdyn model 2.1 – Accelerating point mass [play]

```
y = 10
g = 9.81
m = 0.6
f = 0

# New, automatically created differential equation for position
y’ = "dy"

# New, automatically created state variable for velocity
dy = 0

# Acceleration as differential equation for velocity
dy’ = "-g + f / m"
```

The final implication is that although the example above does not define any edges, differential equations are always necessarily defined on an edge. Recall from section 2.2.1 that every node contains a special edge for which the input and output are set to the node itself. Using the prime syntax inside a node, we have defined a differential equation on the self-edge of that node (in this case the network itself). Figure 2.2 shows the value of `y` over time when simulating this trivial network.

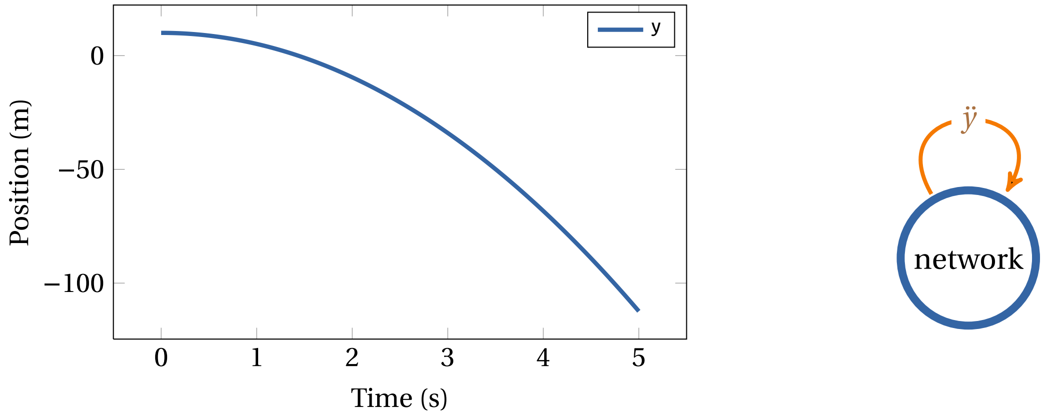


Figure 2.2 – Left: System output of a simple point mass accelerating due to gravity. Right: conceptual representation of a self-edge

2.3.2 Nodes

Nodes become useful when we want group variables (and other nodes) in a self-contained subsystem. Defining nodes in the cȯdyn language is done using the syntax shown in model 2.2.

Just as with the top-level node, we can define variables inside the newly defined nodes. All cdn model 2.2 – Two accelerating point masses with friction [play]

```
# Global gravity, accessible from child nodes
g = 9.81

# Define a single node named n1
node "n1" {
    # Position of the point mass
    y = 20

    # Mass
    m = 0.6

    # External force, air friction
    f = "-dy"

    # Acceleration
    y'' = "-g + f / m"
}

# Define a single node named n2
node "n2" {
    # Position of the point mass
    y = 8

    # Mass
    m = 0.4

    # External force, air friction
    f = "-dy"

    # Acceleration
    y'' = "-g + f / m"
}
```

variables defined as such are locally scoped to the node. Variables from parent scopes can also be accessed, which is shown above by having the nodes access the globally defined gravity variable `g`. Figure 2.3 shows the result of simulating this network.

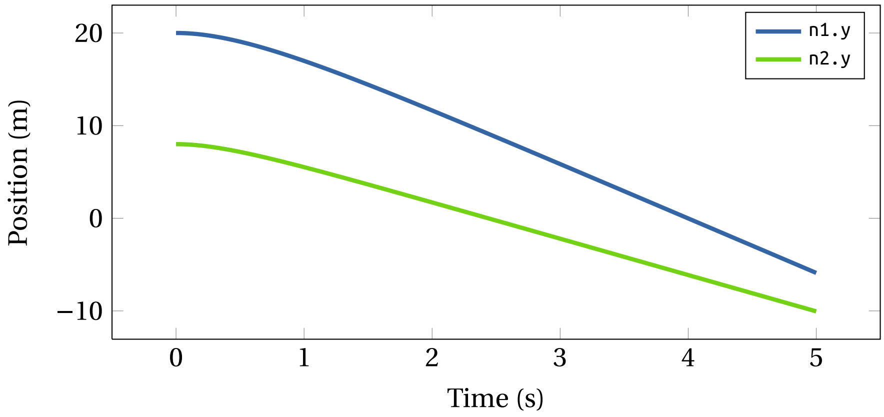


Figure 2.3 – System output of two simple point masses accelerating due to gravity and simple air friction.

2.3.3 Edges

Having defined nodes, we can now implement coupling between **state** variables in different nodes using edges. The syntax for adding an edge can be seen in model 2.3. In this example we take the same network as defined before in model 2.2. We now introduce coupling between the nodes modeling an additional term to the acceleration of both point masses due to a bidirectional spring connecting the two.

The syntax for creating an edge is:

```
edge "name" from "input-node" to "output-node" {
}
```

The name of the edge is optional, and if left out a unique name will be automatically generated for it based on the names of the input and output nodes. When an edge is declared, special attributes can be applied to it making certain constructions easier than they would otherwise be. In model 2.3 we use the `<bidirectional>` attribute to automatically create a reverse edge without having to explicitly define it. Figure 2.5 shows the graphical representation of the network so created.

If we look at the output of this system in figure 2.4 we see that the output starts to already be slightly more interesting. The coupling makes it such that the point masses start to oscillate around each other while accelerating downwards. The simulated air friction cause the oscillations to dampen out, resulting in the two masses accelerating together towards the end of the simulation.

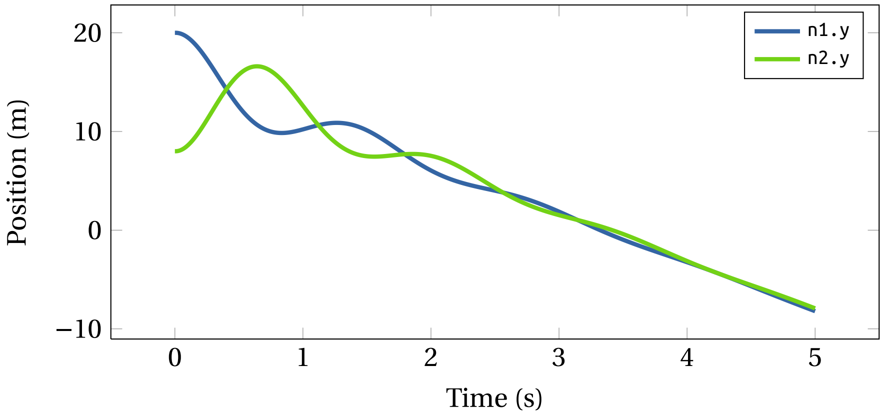


Figure 2.4 – System output of two simple point mass accelerating due to gravity and undergoing forces from air friction. Additionally, the two point masses are coupled by a bidirectional spring.

2.3.4 Generators and selectors

Although we are getting somewhere at this point with simple models, some of the examples already show a certain amount of duplication of effort. Ideally we should be able to declare cȯdγn model 2.3 – Coupling by a spring between two point masses [play]

```
g = 9.81

node "n1" {
    y = 20
    m = 0.6
    f = "-dy"

    y’’ = "-g + f / m"
}

node "n2" {
    y = 8
    m = 0.4
    f = "-dy"

    y’’ = "-g + f / m"
}

# Create a bidirectional edge between the two nodes implementing
# a simple, bidirectional spring, applying a force resulting in
# additional acceleration
<bidirectional>
edge from "n1" to "n2" {
    # Stiffness of the spring. Variables can be defined inside edges
    # as well and allow for convenient definitions of constants and
    # temporary expressions.
    K = "5"

    # Additional acceleration due to the force of the spring. Note
    # that we have to apply the acceleration to the differential
    # equation of the velocity from inside the edge.
    dy’ += "K * (input.y - output.y) / output.m"
}
```

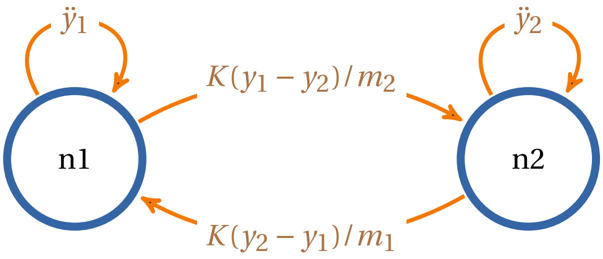


Figure 2.5 – Graphical representation of a network of two coupled point masses.

models in a more concise manner. One of the central concepts in the language that enable this is that of *generators* and *selectors*. These concepts are important because it makes the modeling language both expressive and powerful.

Generators are a special construct in the language which allow to quickly *generate* multiple names at the same time. Generators can be used anywhere in the language where names or identifiers are expected, such as when declaring nodes, edges, variables etc. Generators are always embedded inside strings, and are enclosed by curly braces. When parsed, they are expanded in a combinatorial way, allowing nesting of multiple generators in the same name. In addition to generating multiple names, generators also support embedded calculations, mapping and reducing (see table 2.6).

Table 2.6 – Supported generator syntaxes

Syntax	Description
`"n{1:3}"`	Generate the names `n1, n2, n3`
`"$(expression)"`	Performs an embedded calculation of the provided expression
`"n{1:5|$(@0 * 2)}"`	Make a new generator by mapping each value using an expression. Values can be referred to by `@0`. The result of the provided example would be a generator for the names `n2, n4, n6, n8` and `n10`
`"{1:3||@0 + @1)}"`	Reduce a generator to a single name by successively applying the provided expression, substituting `@0` with the first and `@1` with the second value. The result of the example would be a single literal value `1 + 2 + 3`

Model 2.4 shows the same model as defined in 2.2, this time using generators to generate the two nodes at the same time. As shown, without loss of expressiveness, the model is now more concise while still easy to understand. It is now easier to start modeling more complex systems.

cȯdγn model 2.4 – Basic generator syntax [play]

```
g = 9.81

# Define two nodes at the same time, n1 and n2, by using generator syntax.
# This creates the two nodes in parallel. Definitions inside their scope
# now apply to both nodes at the same time.
node "n{1:2}" {
    # Inside, we can still declare separate values for each node, by
    # using a square bracket syntax
    y = ["10", "8"]

    # Mass
    m = [0.6, 0.4]

    # External force
    f = 0

    # Acceleration. The same for both nodes.
    y’’ = "−g + f / m"
}
```

Model 2.5 shows what we can do now. This example models a system of 10 nodes each with two state variables, `x` and `y`. All nodes get accelerated downwards on `y` due to gravity while a second external force is being exerted. Additionally, the first node on both right and left sides is modified to add a force coming from a spring connected to the fixed frame. Finally, all nodes between the right and left side are coupled bidirectionally, such that a spring acts on both the x and y states, pulling the point masses of the right and left side towards each other. Figure 2.6 shows the output of this system (a quasi-chaotic regime), while figure 2.7 shows the conceptual representation generated by this model definition.

cȯdyn model 2.5 – Generating many nodes and edges using generators [play]

```
g = 9.81

node "n{1:5}{r,l}" {
    m = 1.5

    # Generator syntax also applies to variable names
    "{x,y}" = "rand(−5, 5)"

    x’’ = 0

    # External force implementing damping (due to friction)
    Dy = 0.5
    Fy = "−Dy ∗ dy"

    # Acceleration due to gravity plus external force
    y’’ = "−g + Fy / m"
}

# We can open up existing nodes and modify them. Here
# we will add a simple fixed spring to the first node on
# both left and right sides
node "n1{r,l}" {
    Ky = 10

    # Set external force to damping plus a spring force
    # pulling back the point mass to 0
    Fy = "−Dy ∗ dy + Ky ∗ −y"
}

# This generates full coupling between the left and right
# nodes
<bidirectional>
edge from "n{1:5}r" to "n{1:5}l" {
    Kx = "5"
    Ky = "20"

    dx’ += "Kx ∗ (input.x − output.x) / output.m"
    dy’ += "Ky ∗ (input.y − output.y) / output.m"
}
```

Contexts and expansions

When generators are used, they generate a so-called expansion context which can be accessed within their defining scope. Expansion contexts can be referenced in the cȯdyn language using `@n` syntax, where `n` is a number referring to a specific group in the expansion context. The `0` group is always the full generated name, while groups `1` to `n` refer to curly brace expansions in order of occurrence. Model 2.6 shows the basic usage of referring to expansion contexts.

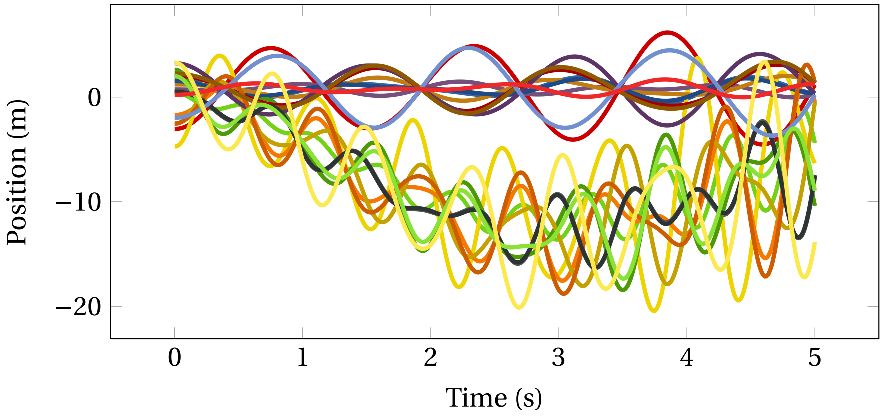


Figure 2.6 – System output of a densely coupled system of point masses.

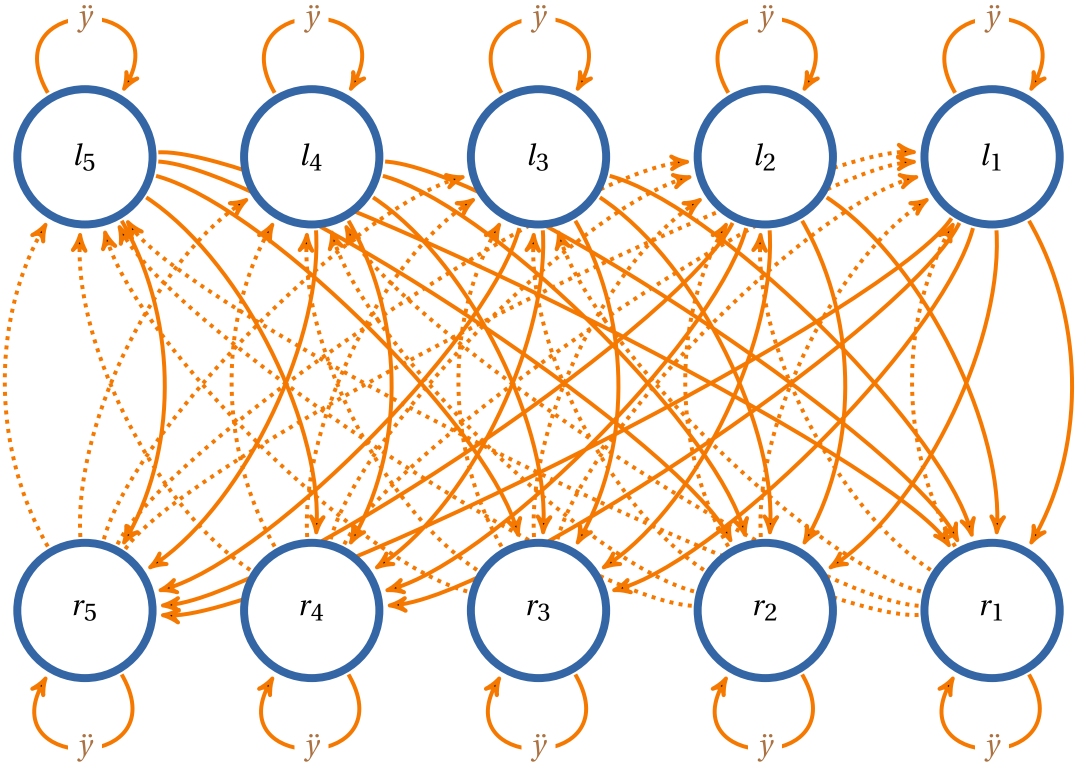


Figure 2.7 – Conceptual rendering of a network of a densely coupled system of point masses. The simple model definition in model 2.5 using generators demonstrates the generative abilities of the cȯdẏn language.

Selectors

Where *generators* allow for the creation of new elements in the model, *selectors* on the other hand allow for referring to one more existing elements in the model using a selection pipeline. They can be used to select a subset of existing elements based on certain criteria (for example cȯdyn model 2.6 – Use of expansion contexts [play]

```
node "n{1:3}" {
    # Here we can access @0 which gives respectively n1, n2 and n3.
    # We also have @1 available, resulting in 1, 2 and 3 respectively

    # Note that since ‘v’ is a name that itself generates a context,
    # we need to access the second level expansion context by using @@
    # in its value to access the 1, 2, 3 expansions from the node name.
    # The $() syntax evaluates an inline calculation which is evaluated by
    # the language parser at compile time.
    v = "$(@@1 * 2)"
}
```

matching name or having a certain variable). Selectors are useful to open existing elements and partially redefine them, or to select input and output nodes for edges.

Selectors are defined as pipelines, where the output from the previous selector serves as input to the next selector. Each element of the pipe line transforms the input it receives to produce a new set of elements. To select on names of elements (for example a node name), *generators* or regular expressions (enclosed in) can be used as pipeline elements. The initial input to the selector pipeline is defined by the current modeling context the selector is used in.

Table 2.7 lists some of the most commonly used selectors. Model 2.7 shows some examples of how these selectors can be used to select nodes based on various selection criteria.

2.3.5 Templates

When there is common functionality to be shared between multiple nodes or edges, templates can be defined and inherited from. This provides a useful abstraction of functionality that can be applied to multiple nodes and allows for building libraries of functionality which can be consumed by various models. Templates are defined in a special block in the network. Model 2.8 shows the basic usage of templates.

2.3.6 Integrator

The type of integrator to use when numerically integrating the network can be specified in a special, top-level `integrator` block. The default integrator is Euler and the default time step is set to 1 millisecond. Model 2.9 shows the basic usage of the `integrator` block.

cȯdyn model 2.9 – Specify the type of integrator [play]

```
integrator {
    # "The" Runge Kutta method, which refers to the 4th order RK.
    method = "runge-kutta"
    default-timestep = "0.01"
}
```

cȯdyn model 2.7 – Basic selector syntax [play]

```
# Define some nodes n1_left to n3_right
node "n{1:3}_{left,right}" {
}

# Open previously defined n1_right to n3_right nodes using a regular expression
node /n(.*)_right/ {
    # Define a variable v on which we can later select
    v = 1
}

# Create an edge between all left and right nodes with matching names
# using a regular expression selector and generator selector
edge from /(.*)_left/
       to "@1_right" {}

# Create a reverse edge for each pair of nodes which already have
# a connecting edge. This has the same effect as using the
# <bidirectional> attribute
edge from nodes | if(inputs) | name
       to @0 | inputs | input {}

# Select nodes which do not yet have a variable named ‘v’ and
# define a variable ‘v’ with a different value in it
node not(children | variables | "v") {
    v = 2
}
```

cȯdyn model 2.8 – Templates [play]

```
templates {
    node "pointmass" {
        m = 1
        f = 0

        y’’ = "-g + f / m"
    }

    edge "spring" {
        K = "5"
        dy’ += "K * (input.y - output.y) / output.m"
    }
}

# Construct 5 nodes inheriting from pointmass
node "n{1:5}" : "pointmass" {}

# Construct edges between neighboring nodes, applying the
# spring template to each edge
<bidirectional>
edge from "n{1:5}" to "n$(@1 + 1)" : "spring" {}
```

2.3.7 Events

When modeling hybrid dynamics, events can be used to switch between different states of the system effectively using condition expressions. In the language, events are expressed by a special block which configures:

Table 2.7 – List of common selectors

Selector	Description
`root`	Select the root network
`children`	Select the direct children of objects in the selection. Children include child nodes and variables
`parent`	Select the parent node of each object in the selection
`first`	Select the first object in the selection
`last`	Select the last object in the selection
`edges`	Filter the selection keeping only edges
`nodes`	Filter the selection keeping only nodes
`variables`	Filter the selection keeping only variables
`input`	Select the input node for each edge in the selection
`output`	Select the output node for each edge in the selection
`inputs`	Select all edges projecting onto each node in the selection
`outputs`	Select all edges projecting from each node in the selection
`name`	Add an expansion context with the name of each object
`has-template(selector)`	Filter objects having the template specified by the provided `selector`
`recurse(selector)`	Recursively apply the specified `selector` to the selection
`if(selector)`	Filter selection keeping only elements for which the provided `selector` results in a *non-empty* set
`not(selector)`	Filter selection keeping only elements for which the provided `selector` results in an *empty* set
`generator`	Filter selection based on element names
`/regex/`	Filter selection based on matching element names to the specified regular expression
`|`	Pipe input from the previous selector to the next
`.`	Shorthand syntax for `| children`

1. In which *event-states* the event is active
2. To which *event-state* the event transitions its containing node
3. The condition for which the event should be activated
4. Whether or not any event refinement should take place
5. Any discrete variable changes to be executed when the event activates

Model 2.10 shows the model specification of simple bouncing pogo-point masses (each point mass is on a virtual spring/damper pogo stick). Each pogo-point mass can be in one of two cȯdyn model 2.10 – Basic usage of events [play]

```
integrator {
    method = "runge−kutta"
}

templates {
    node "pogopoint" {
        initial−state "air"

        pogolength = 0.1
        bounced = 0

        y = 1
        m = 1
        K = 1000
        D = 1

        # Spring force of the ball when it is being compressed
        fspring = "K ∗ (pogolength − y)"

        # Damping force of the ball when it is being compressed
        fdamping = "−D ∗ y’"

        # Acceleration of y due to gravity
        y’’ = "−g"

        # Acceleration of y due to the spring and damping force. This
        # term is only active when the pogo stick is in contact with
        # the ground
        y’’ = "(fspring + fdamping) / m" state "ground"

        # Transfer from the air to the ground when y becomes smaller than
        # the pogo stick length
        event "air" to "ground" when "y < pogolength" within 0.001 {
            # Keep track of the number of times we bounced
            set bounced = "bounced + 1"
        }

        # Transfer from ground to air when y becomes larger than the pogo
        # stick length
        event "ground" to "air" when "y > pogolength" within 0.001 {}
    }
}

g = 9.81

node "p{1:3}" : pogopoint {
    y = "rand(1, 3)"
    m = [0.6, 0.3, 0.4]
    K = [1200, 500, 600]
    D = [1, 1.5, 1.8]
}
```

states, in the air or in contact with the ground. When in the air, the pogo-point mass is only subject to gravitational forces, accelerating it towards the ground. As soon as the pogo stick touches on the ground (i.e. when `y < pogolength`), the state of the node is changed to the `ground` state. In this state, the pogo-point mass is subject to an additional acceleration term due to the spring and damper of the pogo stick.

Figure 2.8 shows the output of this system. As can be seen, the pogo-point masses correctly show damped bouncing trajectories as expected. Errors on the exact time of contact are kept small by the event refinement which refines the timestep until the error of the event condition zero crossing (`y < pogolength`) is smaller than 0.001.

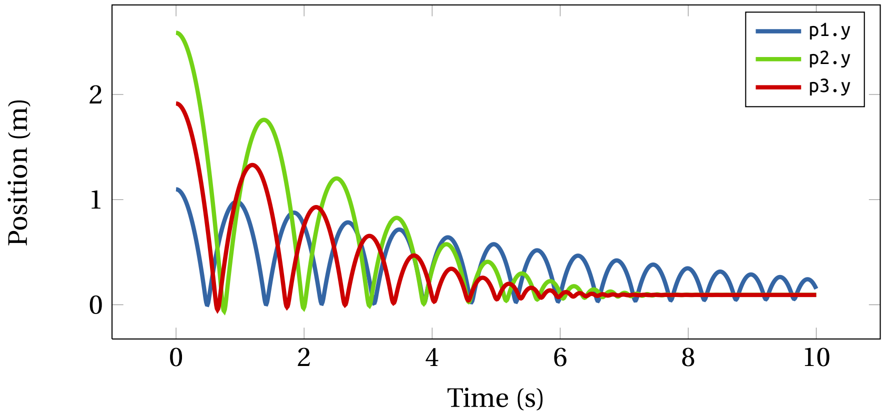


Figure 2.8 – Output of multiple pogo-point mass dynamics with various initial conditions, masses and spring/damper characteristics.

2.4 Example I: Central pattern generators

The examples of systems modeled in cȯdyn shown until now explain individual system features well, but the systems themselves are somewhat contrived. One type of real dynamical systems for which cȯdyn is very suitable as a modeling tool are central pattern generators. "*Central pattern generators (CPGs) are neural circuits found in both invertebrate and vertebrate animals that can produce rhythmic patterns of neural activity without receiving rhythmic inputs*" (Ijspeert, 2008). We will focus here on their mathematical modeling and choose an abstract (instead of a neurological) representation of the CPG. There are a number of well known abstract oscillator models that are widely used to model the dynamics of oscillatory systems with interesting characteristics. For example, the Hopf oscillator is governed by the following differential equations in Cartesian coordinates:

$$\dot{x} = \gamma(\mu - r^2)x - \omega y \tag{2.8}$$

$$\dot{y} = \gamma(\mu - r^2)y + \omega x \tag{2.9}$$

$$r = \sqrt{x^2 + y^2}, \tag{2.10}$$

with **state** variables x and y, angular frequency ω, desired oscillation amplitude $\sqrt{\mu}$ and γ being a constant dictating the speed of convergence to the limit cycle of the oscillator. We can also write down the equations of the Hopf oscillator in Polar coordinates, providing an alternative model for the same system:

$$\dot{\phi} = \omega \tag{2.11}$$

$$\dot{r} = \gamma(\mu - r^2)r, \tag{2.12}$$

with **state** variables ϕ (phase of oscillation) and r (amplitude of oscillation). These two systems are exactly equivalent and only differ in their representation of coordinates to express it. Representations however are important. In Cartesian coordinates, the Hopf oscillator can be modified to be frequency coupled with external oscillatory signals (Righetti and Ijspeert, 2006). On the other hand, the Polar coordinate representation allows for much more straightforward coupling of the phase of two or more oscillators. One commonly used coupling on the phases of oscillators in Polar coordinates is the following:

$$\dot{\phi}_{ij} = w_{ij} r_i \sin(\phi_i - \phi_j - \theta_{ij}), \tag{2.13}$$

where $\dot{\phi}_{ij}$ is the coupling term from oscillator i to j and is added to the differential equation of ϕ_j. w_{ij} is the coupling strength, r_i is the amplitude of oscillator i, ϕ_i and ϕ_j are the phases of respectively oscillator i and j, and θ_{ij} is a phase bias at which the two oscillators should synchronize. Note that this coupling is diffusive (i.e. it disappears when the two oscillators are in synchrony). The coupled system can now be represented as:

$$\dot{\phi}_i = \omega_i + \sum_j \dot{\phi}_{ij} \tag{2.14}$$

$$\dot{r}_i = \gamma_i(\mu_i - r_i^2)r_i \tag{2.15}$$

The system from equations 2.14 and 2.15 can be modeled by writing down the isolated oscillator systems first as a single node with two **state** variables. Then, we introduce edges to couple the individual oscillators. Model 2.11 implements such a model of 5 oscillators with nearest neighbor coupling. The phase bias θ_{ij} is set such that the 5 oscillators combined represent one full traveling wave. The coupling is furthermore symmetric (bidirectional) and consistent (i.e. $\theta_{ij} = -\theta_{ji}$ in this case). Figure 2.9 shows how this system behaves when simulated. As can be seen, the phases of all oscillators start out with random initial conditions, but quickly converge to their desired phase locked behavior.

2.4.1 Van der Pol

Various other popular types of oscillators are easily modeled in cȯdγn as well and provided as part of the standard library of cȯdγn. Model 2.12 shows basic templates for the Van der Pol oscillator. Here it can be seen that cȯdγn also makes it easy to make templates inherit from each other to create new nodes that slightly alter existing functionality. This allows for the creation of families of systems in modular ways.

cȯdyn model 2.11 – Basic network of coupled Hopf oscillators [play]

```
templates {
    node "polar_hopf" {
        f = 1
        omega = "2 ∗ pi ∗ f"

        p’ = "omega"

           mu = 1
        gamma = 5

         r = "mu"
        r’ = "gamma ∗ (mu − r^2) ∗ r"

        x = "r ∗ cos(p)"
    }

    edge "polar_coupling" {
        bias = 0
        weight = 1

        p’ += "weight ∗ input.r ∗ sin(input.p − output.p − bias)"
    }
}

# Define a macro ’n’. Its value can be used later using the @n syntax.
defines {
    n = "5"
}

node "h{1:@n}" : polar_hopf {
    p = "rand(−pi, pi)"
    r = 0.001
}

<bidirectional>
edge from "h{1:@n}" to "h$(@1 + 1)" : polar_coupling {
    s = [−1, 1]
    bias = "s ∗ $(2 ∗ pi / @n)"
}
```

cȯdyn model 2.12 – Van der Pol oscillator templates [play]

```
templates {
    node "van_der_pol" {
        mu = 5

        x = "1"
        x’’ = "mu ∗ (1 − x^2) ∗ x’ − x"
    }

    node "van_der_pol_forced" : "van_der_pol" {
        A = 1

        p’ = "2 ∗ pi"
        x’’ = "mu ∗ (1 − x^2) ∗ x’ − x + A ∗ sin(p)"
    }
}
```

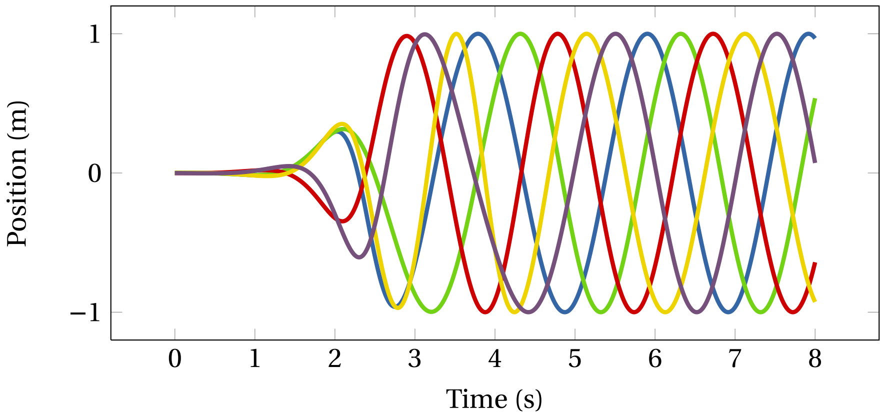


Figure 2.9 – Output of a basic system of phase coupled Hopf oscillators. The 5 oscillators are initially not phase locked, but quickly converge to their desired phase locked behavior

2.4.2 Matsuoka

Another popular oscillator, especially in robotics, is the biologically inspired Matsuoka oscillator. The model for this oscillator resembles a neuronal circuit where oscillation occurs by the mutual inhibition of two neurons. The basic differential equations that govern this system are the following:

$$\tau_1 \dot{x}_1 = c - x_1 - \beta v_1 - \alpha y_2 \tag{2.16}$$

$$\tau_2 \dot{v}_1 = y_1 - v_1 \tag{2.17}$$

$$\tau_1 \dot{x}_2 = c - x_2 - \beta v_2 - \alpha y_1 \tag{2.18}$$

$$\tau_2 \dot{v}_2 = y_2 - v_2 \tag{2.19}$$

$$y_i = \mathtt{max}(x_i, 0) \tag{2.20}$$

$$y = y_1 - y_2 \tag{2.21}$$

To model this oscillator, we can separate the equations and first model the individual neurons without their coupling. We then introduce a bidirectional edge between the two neurons which implements their mutual inhibition. Finally, we can embed this system inside a new node which represents the final Matsuoka oscillator. The oscillator state is calculated in this new node from the output of the two neuron nodes. Model 2.13 shows a basic implementation of this idea.

2.4.3 Morphed nonlinear phase oscillator

One of the difficulties with working with the previously mentioned oscillator dynamical systems is that it is often hard to design them such that they exhibit a desired output pattern. In robotics in particular, oscillators (if used) often drive actuators (for example provide input to a position or torque controller). While the intrinsic properties, such as a stable limit cycle, cȯdyn model 2.13 – Matsuoka oscillator template [play]

```
templates {
    node "neuron" {
          y = "max(x, 0)"

        x = "rand(-0.1, 0.1)"
        v = "rand(-0.1, 0.1)"

        x' = "1 / tau * (c - x - b * v)"
        v' = "1 / Tau * (y - v)"
    }

    node "matsuoka" {
          a = "2"
          b = "2"
          c = "1"
        tau = "0.1"
        Tau = "0.1"

        node "neuron{1,2}" : neuron {}

        <bidirectional>
        edge from "neuron1" to "neuron2" {
            x' += "-1 / tau * a * y"
        }

        x = "neuron1.y - neuron2.y"
    }
}
```

of these oscillators are of interest, it becomes important to be able to accurately control the shape of oscillation as well.

In Ajallooeian *et al.* (2013) we developed a family of nonlinear oscillators with characteristics which make it easy to design oscillators with arbitrary limit cycle shapes. The principal idea is to use a base oscillator with an existing limit cycle, and morph it through the use of a shaping function to a new, desired limit cycle shape. One of the simplest realizations of such an oscillator is to take a simple amplitude controlled phase oscillator as the base of the system, and define a shaping function $f(\phi)$ which provides the desired output signal as a function of the oscillator phase ϕ. The only condition for $f(\phi)$ is that it must be differentiable. Considering the following base oscillator:

$$\dot{\phi}_{\mathbb{B}} = \omega \tag{2.22}$$

$$\dot{r}_{\mathbb{B}} = \gamma(\mu - r_{\mathbb{B}}), \tag{2.23}$$

with $\phi_{\mathbb{B}}$ the base oscillator phase and $r_{\mathbb{B}}$ the base oscillator amplitude, we can write the realization of morphed oscillator, using $f(\phi)$ as a shaping function, as follows:

$$\dot{\phi}_{\mathbb{S}} = \dot{\phi}_{\mathbb{B}} \tag{2.24}$$

$$\dot{r}_{\mathbb{S}} = \mu \dot{f}(\phi_{\mathbb{S}}) + \gamma(\mu f(\phi_{\mathbb{S}}) - r_{\mathbb{S}}) \tag{2.25}$$

Recall from section 2.2.2 that cȯdyn has full support for symbolic derivation, including user defined function. We can therefore implement this system in a straightforward manner as a cȯdyn model. Model 2.14 shows a basic implementation of such a system. It first models the general morphed oscillator as a template, which can then be realized while specifying a user defined function for shaping $r_{\mathbb{S}}$.

cȯdyn model 2.14 – Example model of a morphed phase oscillator [play]

```
templates {
    node "morphed" {
        omega = "2 * pi"
           mu = "1"
        gamma = 1

        # User defined shaping function. This can be overridden
        # in realizations of this template
        f(theta) = "sin(theta)"

        p’ = "omega"
        r’ = "mu * f(p)’ + gamma * (mu * f(p) - r)"
    }

    edge "coupling" {
        bias = 0
        weight = 1
        p’ += "weight * sin(input.p - output.p - bias)"
    }
}

node "m1" : morphed {
    f(theta) = "cos(theta * 2 + 0.2 * pi) * (0.1 + sin(theta))"

    p = "rand(-pi, 0)"
    r = 6
}

node "m2" : morphed {
    f(theta) = "cos(theta * 2 + 0.2 * pi) * (0.1 + sin(theta)) + 0.5"

    p = "rand(0, pi)"
    r = -4
}

<bidirectional>
edge from "m1" to "m2" : coupling {
    weight = 0.5
}
```

2.5 Example II: SLIP model

The Spring Loaded Inverted Pendulum model is a very well known and extensively researched model which describes fundamental properties of running (Seyfarth *et al.*, 2002). The basic model is relatively simple, consisting of a single leg, modeled as a point mass which undergoes forces exerted on it from a (preloaded) spring when the (virtual) leg is in contact with the ground. During the swing phase, the point mass undergoes a purely ballistic motion. A

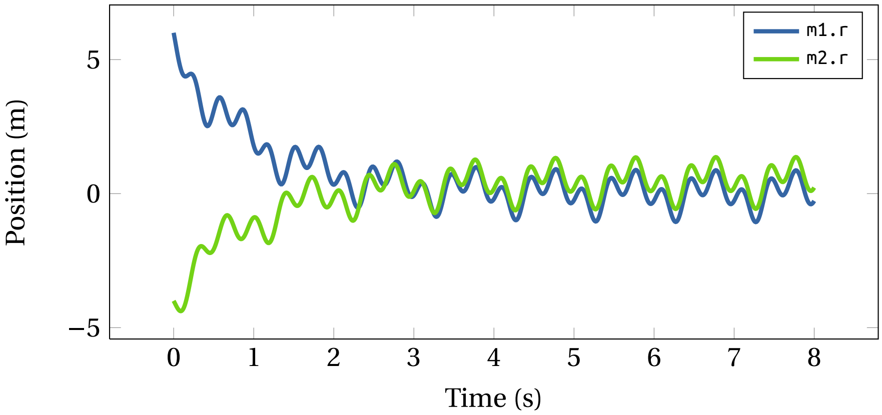


Figure 2.10 – Output of a coupled system of two morphed oscillators with arbitrary shaping functions. The two oscillators start outside their limit cycle but quickly converge. At the same time, a bidirectional coupling on the phase ensures phase locking behavior after a short period of time.

parameter of the model, the *angle of attack*, determines when the leg touches the ground during the ballistic motion and with which angle the leg touches down. The dynamics of the leg during this phase (i.e. swinging the leg forward) are ignored in this model (i.e. the leg is massless). Figure 2.11 depicts a schematic version of the model.

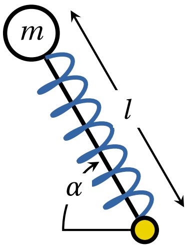


Figure 2.11 – Schematic depiction of the SLIP model. The basic SLIP model consists of a single point mass m connected to a massless spring with rest length l. The angle of attack α determines at which angle the leg transitions from the swing phase to the stance phase during locomotion.

Even though the model is simple, it can be shown to be self-stabilizing and can be used as a basis for deriving motion for more complicated legged structures. The model dynamics for this system can be easily implemented in ċodẏn using events to switch between the hybrid dynamics of the stance and swing phase. Model 2.15 provides the basic model, using default parameters for the angle of attack ($\alpha = 68°$), rest length ($l = 1$), mass ($m = 80$) and spring stiffness ($k = 20$ kN) obtained in (Seyfarth *et al.*, 2002).

As an example, we will replicate the results obtained from (Seyfarth *et al.*, 2002) where the number of *steps-to-fall* is obtained as a function of spring stiffness and angle of attack. We leave all other parameters fixed ($l = 1$, $m = 80$) and vary the stiffness from 0 to 50 kN, and the angle of attack from 40° to 80°. We then forward simulate the model for each combination cȯdyn model 2.15 – Spring loaded inverted pendulum [play]

```
integrator {
    method = "runge−kutta"
}

g = 9.81

node "slip" {
    initial−state "air"

      l = 1                         # Spring rest length
    aoa = "(90 − 68) / 180 ∗ pi"    # Angle of attack
      m = 80                        # Mass
      k = "20000"                   # Spring stiffness

      x = 0
      y = l

     dx = 5
     dy = 0

    x’’ = 0
    y’’ = "−g"

    xc = 0 # Ground contact position. Updated from events.

    leglength = "hypot(x − xc, y)"    # Current leg length
      fspring = "k ∗ (l − leglength)" # Force of the spring
           aol = "atan2(xc − x, y)"   # Current angle of attack

    steps = 0 # Number of steps taken

    # Transition from air to ground on leg touch down
    event "air" to "ground" when "y < l ∗ cos(aoa)" within 0.001 {
        set xc = "x + l ∗ sin(aoa)" # New contact position
        set steps = "steps + 1"
    }

    # Termination condition, falling or more than 24 steps made
    event any to terminate when "y < 0 || steps >= 24" {}

    # Transition from ground to air when spring is fully extended
    event "ground" to "air" when "leglength > l" within 0.001 {}

    # x’’ = projected(FSpring, X) / m
    dx’ = "(−sin(aol) ∗ fspring) / m" state "ground"

    # y’’ = projected(FSpring, Y) / m
    dy’ = "(cos(aol) ∗ fspring) / m" state "ground"
}
```

of stiffness and angle of attack and record for each simulation the number of steps made. Note that the simulations automatically terminate whenever the solution fails (falls down) or when a maximum number of 24 steps has been reached. Figure 2.12 shows the obtained characteristic J-shaped result from running this simulation.

We performed the simulation on a grid of 100 stiffness and 100 angle of attack values, using a Runge Kutta order 4 numerical integrator with a timestep of 1 millisecond. The simulations

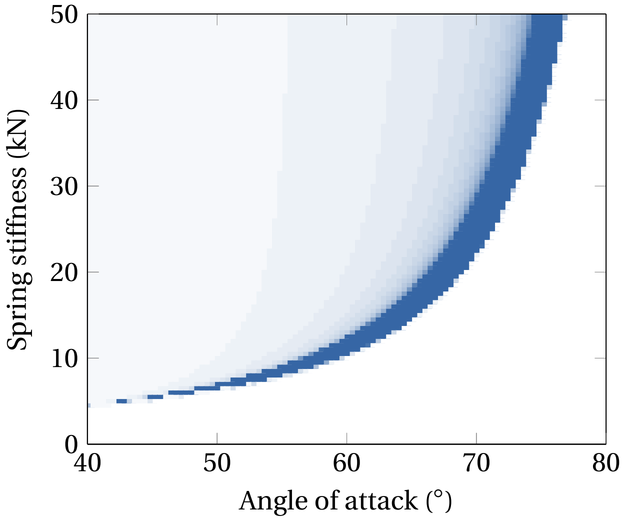


Figure 2.12 – Number of steps to fall for the SLIP model for different combinations of spring stiffness and angle of attack using cȯdγn. The characteristic J-shape indicates the stable region of forward locomotion (Seyfarth *et al.*, 2002).

were run on an iMac, 3.2 GHz Intel Core i3 with 4GB of 1333 MHz DDR3 memory, using codyn 3.6. Evaluating the entire space (i.e. 10000 simulations) took 4 minutes and 38 seconds of real time with libcodyn, averaging at 0.028 seconds per simulation. In section 2.7 we show in more detail how we can obtain better performance by developing a special tool to automatically generate optimized code. Using that tool, we can drastically reduce the total simulation time to only 6 seconds (i.e. an increase in performance of a factor 45).

2.6 Rigid body dynamics

In the previous sections, most of the physically based examples that were shown were very simplified versions of real physical systems. It is relatively simple to derive the dynamical system models for these simplified cases by hand. However, as soon as we start to properly model inertial dynamics and go from single point masses to articulated rigid bodies, the system equations become unwieldy very quickly. Even if the differential equations for simple systems, for example a double pendulum, *can* be derived by hand, the procedure is error prone and leads to enormous equations which are very hard to maintain.

When we look at the equations of motion for rigid body dynamics, the general formulation of the dynamics is given by:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{2.26}$$

This equation is written in terms of state variables $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, which are the *generalized* coordinates of the system. The term *generalized* refers to the fact that one is free to choose any set of coordinates $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ which fully describe the system, and write the equations of motion in terms of these coordinates. For example, one can describe the equations of motion for an N-degree-of-freedom manipulator in terms of their *relative* joint angles or their *absolute* joint angles. The resulting models will exhibit exactly the same behavior, but the exact equations of motion will differ.

In equation 2.26, $\boldsymbol{H}(\boldsymbol{q})$ is called the mass matrix and $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ contains all the forces due to gravity, coriolis effect and centrifugal effects. $\boldsymbol{\tau}$ is the set of *generalized* forces that affect the system. The exact meaning of these forces depends on the choice of generalized coordinates.

The goal of any RBD simulator is then to calculate $\boldsymbol{H}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ given the RBD model specification and the current state of the system. Once obtained, equation 2.26 can be used to either solve for $\ddot{\boldsymbol{q}}$ given $\boldsymbol{\tau}$, or to solve for $\boldsymbol{\tau}$ given $\ddot{\boldsymbol{q}}$. The first refers to solving the *forward* dynamics and is used to simulate the system forward in time, integrating accelerations and velocities to obtain motions. The second refers to *inverse* dynamics in which case all the motions are known, and one is interested in knowing which generalized forces ($\boldsymbol{\tau}$) would cause these motions to occur.

ċodyn has a very unique view on the construction of the equations of motion for rigid body dynamics. It does not treat the rigid body dynamics differently from any other dynamical system, and modeling results in a natural treatment of dynamics using ċodyn concepts. The main motivations and objectives for providing RBD simulations in ċodyn are the following:

1. *Open*: ċodyn is completely open and free to use. There are no limitations to using or modifying it. Models can be easily shared and obtained results replicated by anyone.
2. *General*: the rigid body dynamics are derived generally, and should not impose restrictions of the type of articulated systems which can be modeled and simulated. It is as easy to simulate robotic manipulators as it is to model legged articulated systems.
3. *Extensible*: an important objective for ċodyn is to easily allow extending of the derived equations of motion. The *openness* and *generality* objectives support this, but ċodyn goes a step further due to the way it treats the rigid body dynamics as any other dynamical system. This makes it straightforward to add customized joint models, contact models or additional dynamics.
4. *Expressive*: a recurring motivation for ċodyn is to be expressive, without loss of performance. Where existing simulators need special purpose, inaccessible (but high performance) RBD simulation engines, ċodyn expresses all dynamics, without exceptions, in the ċodyn language. Equations are therefore written close to their mathematical, textbook form and easy to understand.
5. *Fast*: ċodyn explicitly aims to be a fast simulator. Deriving and simulating the RBD equations of motion in a straightforward and naive way results in very slow simulation times. Not only does ċodyn provide the automatic derivation of fast, optimized code, it importantly does so without loss of generality or expressiveness in the language. Section 2.7 explains in detail how ċodyn manages this.

Before we dive into the details of the cȯdγn approach to RBD, we first briefly overview existing state of the art RBD simulators.

2.6.1 Existing simulators

There are generally speaking two popular, but different methods for solving rigid body dynamics. Although both methods are derived from the same laws of mechanical physics, they greatly differ in the way that they solve for the equations of motion. The first method (referred to here as *constraint-solver RBD simulators*), which has been popularized through simulators originally developed for games (such as ODE, Bullet Physics or Box 2D) is based on the direct application of Newton's second law for linear motion and Euler's second law for angular motion. They are usually first order simulators (velocity based) which describe the motions of single bodies independently, and then continue to explicitly add (and solve for) constraints imposed by joints. This method is in stark contrast to the second popular method (*generalized EoM RBD simulators*), which is based on the derivation of the equations of motion in generalized coordinates. Here, the constraints imposed by the joints are solved for by describing the dynamics in terms of coordinates that include these constraints implicitly. The actual constraint forces therefore do not need to be solved for explicitly which leads to very accurate and stable simulations. Furthermore, having the equations of motion explicitly derived, simulators based on this method can be used for analysis, control and design.

There has been a recent interest in the development of accurate and importantly *open* RBD simulators which has lead to several new software packages specifically targeting robotics and research. Some of the important recent developments are briefly discussed here.

ODE/Bullet derivatives

There are many available software packages that use ODE (Open Dynamics Engine) or Bullet Physics as the underlying library to solve for the equations of motion. Both ODE and Bullet (which itself is a derivative of ODE) are *constraint solver RBD simulators*. There are many advantages to this method. Simulations are generally fast, since joints are modeled as constraint equations which can be solved for numerically very quickly. The method also does not require computing global system entities, such as the mass matrix, since computations for each body are done locally and constraints solved for explicitly. Models can therefore also be constructed very quickly, since no complicated derivations have to be performed to obtain the equations of motion in a reduced, symbolic form.

There are however certain disadvantages of these simulators which make them largely unsuitable for scientific purposes. In particular, since these simulators were designed for gaming purposes, they allow for certain non-physical phenomena with the sole purpose of stabilizing the simulation. It thus trades accuracy for speed and stability which can lead to unexpected results. One known issue is for example that joints can easily drift apart during simulation due to numerical inaccuracies while solving constraints. This problem is addressed by constraint force mixing techniques that try to stabilize this drift. Nevertheless, this can result in joints temporarily being separated. It also means that energy can be easily injected into the system as a side effect. Constraint forces (and by implication contact forces) are therefore unreliable as estimations of actual physical forces. They represent a solution to the system constraints but are otherwise non-physical. Finally, since the equations of motion of the full system are never actually derived, these simulators do not provide any system kinematic (e.g. Jacobians) or inverse dynamics models and are thus only suitable to simulate forward dynamics.

Because of these reasons, simulators such as ODE and Bullet are a great tool when the sole interest is to forward simulate systems, i.e. a reasonable realization of RBD to obtain physical motions and general model validations. They are however a poor solution when one is interested in deriving dynamics (or kinematics) based control laws, and particularly ill fitted to perform accurate simulations, derive design principles (e.g. minimum bearing specifications) or estimate interaction forces.

It should be noted that in a recent development (late 2013), acknowledging the need for more accurate simulations, in particular for robotics, the Bullet physics engine has developed support for a special purpose engine based on Featherstone's articulated rigid body algorithm. Bullet is therefore no longer solely an excellent tool for introducing RBD in games, but is interesting also as a tool for research and in particular, robotics research.

The following simulators are all *generalized EoM RBD* simulators rather than *constraint-solving RBD* simulators.

ADAM

ADAM is a sophisticated, proprietary, multibody dynamics simulator developed by Msc software. It provides state of the art tools for system design, excelling at extremely detailed and accurate simulations. As an RBD simulator, it is an industry standard and used by for example large car manufacturers to aid in the design and analysis of complicated mechanical systems. Designed for the industry, it is however not commonly used for Robotics research. Not only is it a closed product, simulations are also very slow, due to their great level of detail, and are focused specifically on industrial design.

OpenSim

Another very interesting and promising dynamics simulator is OpenSim/SimTk (Delp *et al.*, 2007). Developed at Stanford, it largely focuses on the simulation of musculoskeletal models used to study biomechanics and rehabilitation. Its primary target is to study human motion, but it can be used for humanoid robotics as well.

SimMechanics

SimMechanics is a popular, simulink based simulator developed by Matlab. It is particularly attractive when the user is already familiar with development of models in Simulink, in which case modeling of mechanics can be done just as any other system. Of course, Matlab is not freely available, but it is probably one of the most professional and furthest developed software package for mechanical simulation used in Academia. Models can be either simulated directly in Simulink, or compiled to native code which accelerates the simulation time.

Robotran

Robotran is an RBD simulator developed by the university of Louvain which uses a symbolic derivation of the equations of motion based on a recursive Newton Euler formulation (Samin, 2003). Interestingly, it allows for modeling of multiple domains, most notably the mechanical and electrical domains. It can be used for example to model both the rigid body dynamics and the motor actuator dynamics in a single, consistent model. Unfortunately, the current version of Robotran is relatively slow, and depends both on the use of Matlab and relies on an external server to derive the equations of motion. It can therefore not be used freely. It also supports only a limited number of joint models (only single DOF rotational and prismatic joints are supported) and does not provide any sophisticated ground contact models.

MuJoCo

MuJoCo (Todorov *et al.*, 2012) is a very recent and promising RBD simulator. It aims to be a general purpose, fast and accurate dynamics solver, with a particular focus on deriving control from the rigid body dynamics. It derives equations of motion in general coordinates and has a state of the art constraint solver to solve for contact dynamics as well as additional, user defined constraints. Unfortunately, although initially advertised as being released as free and open software, it has not been made available at the time of writing.

cȯdyn

The philosophy behind cȯdyn is quite different from most other simulators. From the beginning, cȯdyn has been a general purpose, coupled dynamical systems framework. It focuses not on algorithms in the traditional sense, but instead relies on declarative modeling. Unlike in most other simulators, which support declaring only model structure, cȯdyn also declares all computation. This does impose some limitations, since if a problem cannot be solely declared, it cannot be represented in cȯdyn. However, it also forces to look at rigid body dynamics from a different perspective.

cȯdyn does not contain any special purpose rigid body dynamics engine. Every and all dynamics are simply declared in terms of physical quantities. There are no additional solvers, nor exceptions made to be able to implement rigid body dynamics. Furthermore, the derivation of the equations of motion is done entirely using the cȯdγn language. The principle is that the user creates a model description, specifying all necessary quantities such as masses, inertia tensors, joint types, etc. Hereafter, a cȯdγn file which gets included extracts structure, through the use of selectors, and completes the dynamics equations by declaring edges to project quantities in the appropriate locations. The result is a fully declarative dynamics model of the rigid body dynamics. There are two main advantages, 1) dynamics are considered as a whole, whether it is rigid body, oscillators or any other dynamics that can be represented in cȯdγn, and 2) a declarative model allows for rigorous optimizations and high performance difficult to obtain otherwise.

The remainder of this section provides a detailed overview of the state of the art rigid body dynamics of cȯdγn.

2.6.2 Deriving equations of motion

There are several different methods for deriving the equations of motion in the form of equation 2.26. Since they derive the same equation, different methods result in the same dynamics, but may differ in computational complexity and numerical stability. Two widely used methods are the Lagrange II and Projected Newton Euler methods.

Lagrange II

The Lagrange II method is based on the relation between the motions of a system and its kinetic and potential energies. It is a popular method because derivation of the equations is relatively simple. The general procedure is shown in equations 2.27 to 2.30.

$$\boldsymbol{T} = \sum_i \frac{1}{2} m_i \dot{\boldsymbol{r}}_i^T \dot{\boldsymbol{r}}_i + \frac{1}{2} \boldsymbol{I}_i \boldsymbol{\Omega}_i^T \boldsymbol{\Omega}_i \tag{2.27}$$

$$\boldsymbol{V} = \sum_i -\boldsymbol{r}_i^T \boldsymbol{f}_i \tag{2.28}$$

$$\boldsymbol{L} = \boldsymbol{T} - \boldsymbol{V} \tag{2.29}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \boldsymbol{L}}{\partial \dot{\boldsymbol{q}}}\right) = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{q}} \tag{2.30}$$

Given a multi body, articulated, rigid body system, derive equations for the kinematic ($\boldsymbol{T}$) and potential ($\boldsymbol{V}$) energy at the center of mass ($\boldsymbol{r}_i$) for each body in the inertial frame. Furthermore, $\dot{\boldsymbol{r}}_i$ is the linear velocity of body i and $\boldsymbol{\Omega}_i$ is its corresponding rotational velocity. $\boldsymbol{f}_i$ is the total external force (including gravity) on body i, applied at its center of mass. Then, the *Lagrangian* ($\boldsymbol{L}$) is defined by $\boldsymbol{L} = \boldsymbol{T} - \boldsymbol{V}$ and Lagrange's equation, stated in 2.30 defines the equations of motion. This equation can be rewritten to obtain the $\boldsymbol{H}$ and $\boldsymbol{C}$ matrices in the general form.

The elegance of this method is that it is easy to write down the potential and kinetic energies in the system. The issue however is that to resolve the actual equations of motion, many partial derivatives have to be obtained. Although taking a partial derivative is not particularly hard, problems arise due to the fact that equations quickly become extremely large (and sparse). This in turn requires sophisticated symbolic simplifications to make the resulting equations computationally tractable. Lagrange II is therefore often only used for relatively small systems, or derived once for single models which are not subject to change.

Projected Newton Euler

The Projected Newton Euler method is in many ways an answer to the issues with the Lagrange II method. Instead of computing the derivatives of energy functions, it is based on projecting the Newton-Euler method into generalized coordinates. This avoids computation of expensive partial derivatives while at the same time removing the explicit acceleration constraints that are usually introduced in the standard Newton Euler method to model articulated joints. The general equations of motion derived this way are shown in equations 2.31 and 2.32.

$$\boldsymbol{H}(\boldsymbol{q}) = \sum_i (m_i \boldsymbol{J}_{S_i}^T \boldsymbol{J}_{S_i} + \boldsymbol{I}_i \boldsymbol{J}_{R_i}^T \boldsymbol{J}_{R_i}) \tag{2.31}$$

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_i (m_i \boldsymbol{J}_{S_i}^T \dot{\boldsymbol{J}}_{S_i} \dot{\boldsymbol{q}} + \boldsymbol{I}_i \boldsymbol{J}_{R_i}^T \dot{\boldsymbol{J}}_{R_i} \dot{\boldsymbol{q}}) \tag{2.32}$$

Here J_{S_i} is the *center of mass* Jacobian of body i, J_{R_i} is the *rotation* Jacobian and I_i is the inertia tensor. It may look as if this equations has not gained us much. After all, we appear to still be required to obtain a large number of Jacobians (partial derivatives) and their derivatives. As it turns out however, we can do away with explicitly computing all the Jacobians and instead use recursive methods to implicitly construct them, making this a very computationally efficient method. Almost all software packages which explicitly derive equations of motion are based on a form of projected, recursive Newton Euler.

The derivation of the equations of motion in cȯdγn are based on Featherstone's method (Featherstone, 2008), which is widely regarded as the current state of the art method. The remainder of the section will describe in detail how this method is implemented in cȯdγn, how joint models can be defined and forward dynamics derived, and finally how models can be specified in the cȯdγn language. Note that unless otherwise noted, formulations and equations are adapted from (Featherstone, 2008) and provided for completeness.

2.6.3 Spatial vector algebra

To be able to follow the algorithms in this section, it is necessary to briefly introduce spatial vector algebra, which is used in the formulation of the equations of motion. In spatial vector algebra, motions and forces are expressed in 6D, and it defines the appropriate spatial operations. It usually is more common to express the linear and angular dynamics separately, leading to writing equations in 3D spaces. Instead, spatial vector algebra allows to write equations for both linear and angular dynamics at the same time. Although considered less intuitive, once one grasps the elementary concepts, the resulting equations turn out to be much simpler.

Spatial motion vector

A spatial motion vector, $\hat{\boldsymbol{v}}$ is a vector spanning the 6D motion space, i.e. it is an element of M^6 (the vector space of spatial motion vectors). Although it is possible to choose any vector basis which spans the correct space, the most straightforward choice is to use Plücker coordinates, resulting in $\hat{\boldsymbol{v}} = [\omega_x, \omega_y, \omega_z, v_{O_x}, v_{O_y}, v_{O_z}]^T$. Here ω_x, ω_y and ω_z are angular velocities around the coordinate axis unit vectors. Correspondingly, v_{O_x}, v_{O_y} and v_{O_z} are the linear velocities at the origin of the coordinate frame, along its principal axis. Note that we use the same convention as in (Featherstone, 2008), placing angular velocities first, then linear velocities. This choice is arbitrary as long as it is used consistently and different conventions are in use.

Spatial force vector

Similar to spatial motion vectors, a spatial force vector, $\hat{\boldsymbol{f}}$ spans F^6, the vector space of spatial force vectors. Again choosing the basis vectors for Plücker coordinates, we obtain the spatial force vector as $\hat{\boldsymbol{f}} = [n_{O_x}, n_{O_y}, n_{O_z}, f_x, f_y, f_z]^T$ with n_{O_x}, n_{O_y} and n_{O_z} being forces resulting in rotation (e.g. torques) and f_x, f_y, f_z being linear forces along the unit axes of the frame. Note that F^6 is the vector dual space of M^6 and vice versa.

Spatial transformations

A general transformation (rotations and translations) of a spatial motion vector can be done using a spatial transformation matrix. If we define two coordinate frames A and B, then the transformation ${}^B\boldsymbol{X}_A$ of a spatial motion vector ${}^A\boldsymbol{m}$ (in A coordinates) from frame A to frame B is given by:

$$ {}^B\boldsymbol{m} = {}^B\boldsymbol{X}_A {}^A\boldsymbol{m} \tag{2.33} $$

$$ {}^B\boldsymbol{X}_A = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix} \tag{2.34} $$

Here, $\boldsymbol{E}$ is a 3-by-3 rotation matrix, $\boldsymbol{I}$ is a 3-by-3 identity matrix, $\boldsymbol{0}$ is a 3-by-3 zero matrix and $\boldsymbol{r}$ is a 3-by-1 translation vector. $\boldsymbol{r}\times$ denotes the skew symmetric matrix of $\boldsymbol{r}$, which is defined as:

$$ \boldsymbol{r}\times = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{2.35} $$

It is useful to understand the meaning of the skew symmetric matrix as the matrix form of the cross product. Additionally, recall that the linear velocity $\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t}$ of a vector $\boldsymbol{r}$ resulting from an angular velocity $\boldsymbol{\omega}$ is given by:

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \boldsymbol{\omega} \times \boldsymbol{r} \tag{2.36}$$

This can be extended to spatial vectors. The derivative operator for a motion vector $\hat{\boldsymbol{m}}$ given a spatial velocity $\hat{\boldsymbol{v}}$ then is $\hat{\boldsymbol{v}} \times \hat{\boldsymbol{m}}$, with

$$\hat{\boldsymbol{v}}\times = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} \times = \begin{bmatrix} \boldsymbol{\omega}\times & \mathbf{0}^{3\times3} \\ \boldsymbol{v}_O\times & \boldsymbol{\omega}\times \end{bmatrix} \tag{2.37}$$

Similarly, the derivate operation for a spatial force $\hat{\boldsymbol{f}}$ by a spatial velocity $\hat{\boldsymbol{v}}$ is $\hat{\boldsymbol{v}} \times^* \hat{\boldsymbol{f}}$, with

$$\hat{\boldsymbol{v}}\times^* = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} \times^* = \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{v}_O\times \\ \mathbf{0}^{3\times3} & \boldsymbol{\omega}\times \end{bmatrix} = -(\hat{\boldsymbol{v}}\times)^T \tag{2.38}$$

$$\tag{2.39}$$

Spatial inertia

The spatial inertia of a rigid body defines how the spatial velocity of that body relates to its spatial momentum, and the relationship is given by

$$\boldsymbol{h} = \boldsymbol{I}\boldsymbol{v}, \tag{2.40}$$

with $\boldsymbol{h}$ the spatial momentum and $\boldsymbol{I}$ the spatial inertia. Without going into the derivation, the spatial inertia $\boldsymbol{I}$ is defined by the mass m of the rigid body, its center of mass $\boldsymbol{c}$ and finally its Cartesian inertia tensor $\boldsymbol{I}_C$ at $\boldsymbol{c}$. The resulting spatial inertia can be obtained by:

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{I}_C + m\boldsymbol{c}\times\boldsymbol{c}\times^T & m\boldsymbol{c}\times \\ m\boldsymbol{c}\times^T & mI^{3\times3} \end{bmatrix} \tag{2.41}$$

Spatial acceleration

The spatial acceleration of a body is simply the derivative of the spatial velocity.

$$\hat{\boldsymbol{a}} = \frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{v}} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\boldsymbol{r}} - \boldsymbol{\omega}\times\dot{\boldsymbol{r}} \end{bmatrix}, \tag{2.42}$$

with $\ddot{\boldsymbol{r}}$ the linear acceleration and $\dot{\boldsymbol{\omega}}$ the angular acceleration of the body.

Spatial operations

There are some other useful operations on spatial vectors and spatial transformations. A summary of these operations is given in table 2.8.

Table 2.8 – Commonly used spatial operations

Quantity	**Expression**	
General transform Translation $\boldsymbol{r}$ Rotation $\boldsymbol{E}$	${}^{B}\boldsymbol{X}_{A}$	$\begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}$
Spatial inverse	${}^{A}\boldsymbol{X}_{B}$	$= {}^{B}\boldsymbol{X}_{A}{}^{-1} = \begin{bmatrix} \boldsymbol{E}^{T} & \boldsymbol{0} \\ (-\boldsymbol{E}\boldsymbol{r}\times)^{T} & \boldsymbol{E}^{T} \end{bmatrix}$
Spatial translation	$\mathrm{Xtr}(r)$	$= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \boldsymbol{I} \end{bmatrix}$
Spatial rotation	$\mathrm{Xrot}_{x,y,z}(\theta)$	$= \begin{bmatrix} \boldsymbol{E}_{x,y,z}(\theta) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}_{x,y,z}(\theta) \end{bmatrix}$
$\boldsymbol{E}_{x}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}(\theta) & \mathrm{s}(\theta) \\ 0 & -\mathrm{s}(\theta) & \mathrm{c}(\theta) \end{bmatrix}, \boldsymbol{E}_{y}(\theta) = \begin{bmatrix} \mathrm{c}(\theta) & 0 & -\mathrm{s}(\theta) \\ 0 & 1 & 0 \\ \mathrm{s}(\theta) & 0 & \mathrm{c}(\theta) \end{bmatrix}, \boldsymbol{E}_{z}(\theta) = \begin{bmatrix} \mathrm{c}(\theta) & \mathrm{s}(\theta) & 0 \\ -\mathrm{s}(\theta) & \mathrm{c}(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Spatial quaternion	$\mathrm{Xquat}(\boldsymbol{q})$	$= \begin{bmatrix} \boldsymbol{E}_{q}(q) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}_{q}(q) \end{bmatrix}$
$\boldsymbol{E}_{q}(q)$		$= 2 \begin{bmatrix} q_1^2 + q_2^2 - 0.5 & q_2 q_3 + q_4 q_1 & q_2 q_4 - q_3 q_1 \\ q_2 q_3 - q_4 q_1 & q_1^2 + q_3^2 - 0.5 & q_3 q_4 + q_2 q_1 \\ q_2 q_4 + q_3 q_1 & q_3 q_4 - q_2 q_1 & q_1^2 + q_4^2 - 0.5 \end{bmatrix}$
Affine transform	${}^{B}\boldsymbol{T}_{A}$	$= \begin{bmatrix} \boldsymbol{E} & -\boldsymbol{E}\boldsymbol{r} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}$
Extract $\boldsymbol{E}$	$\mathrm{Xrot3}(\boldsymbol{X})$	$= \begin{bmatrix} \boldsymbol{X}_{11} & \boldsymbol{X}_{12} & \boldsymbol{X}_{13} \\ \boldsymbol{X}_{21} & \boldsymbol{X}_{22} & \boldsymbol{X}_{23} \\ \boldsymbol{X}_{31} & \boldsymbol{X}_{32} & \boldsymbol{X}_{33} \end{bmatrix}$
Extract $-\boldsymbol{E}\boldsymbol{r}$	$\mathrm{Xtr3}(\boldsymbol{X})$	$= - \begin{bmatrix} \boldsymbol{X}_{5,1\ldots3}\, \boldsymbol{E}^{T}_{1\ldots3,3} \\ \boldsymbol{X}_{6,1\ldots3}\, \boldsymbol{E}^{T}_{1\ldots3,1} \\ \boldsymbol{X}_{4,1\ldots3}\, \boldsymbol{E}^{T}_{1\ldots3,2} \end{bmatrix}$
Transform point	$\mathrm{XtrP}(\boldsymbol{X}, \boldsymbol{p})$	$= \mathrm{Xrot3}(\boldsymbol{X}) \cdot \boldsymbol{p} + \mathrm{Xtr3}(\boldsymbol{X})$

2.6.4 Joint models

A joint represents a kinematic constraint by which two bodies are joined together. If we can define this kinematic constraint in a general manner and express its velocities and accelerations that are transmitted from one body to the next over the joint, then we can easily model any type of joint (e.g. revolute, prismatic, cylindrical or spherical).

Let $\boldsymbol{v}_i$ be the spatial velocity of body i and $\boldsymbol{v}_{\lambda(i)}$ be the velocity of the parent (λ_i) of body i, i.e. the body to which the joint connects body i. The velocity across the joint, $\boldsymbol{v}_{J_i}$, is then simply

$$\boldsymbol{v}_{J_i} = \boldsymbol{v}_i - \boldsymbol{v}_{\lambda(i)} \tag{2.43}$$

Equivalently, the joint velocity can be expressed as a function of the generalized velocity of that joint by

$$\boldsymbol{v}_{J_i}(\dot{\boldsymbol{q}}) = \boldsymbol{S}_i \dot{\boldsymbol{q}} \tag{2.44}$$

Here $\boldsymbol{S}_i$ is called the joint *motion subspace*, which is a matrix projecting the generalized velocity $\dot{\boldsymbol{q}}$ to a spatial velocity and is specific to a particular joint type. Note that if we have defined $\boldsymbol{v}_{J_i}(\dot{\boldsymbol{q}})$, then we can obtain the corresponding motion subspace by

$$\boldsymbol{S}_i = \frac{\partial \boldsymbol{v}_{J_i}}{\partial \dot{\boldsymbol{q}}} \tag{2.45}$$

If the motion subspace not only depends on generalized velocities, but also generalized coordinates, then it is necessary to compute a so-called bias velocity product

$$\boldsymbol{c}_{J_i} = \frac{\partial \boldsymbol{S}_i}{\boldsymbol{q}} \dot{q}^T \dot{q} \tag{2.46}$$

The joint *motion subspace* also projects spatial forces, by means of

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^T \boldsymbol{f}_{J_i} \tag{2.47}$$

To obtain the spatial joint acceleration, we can simply differentiate the spatial velocity to obtain

$$\boldsymbol{a}_{J_i} = \dot{\boldsymbol{v}}_{J_i} = \dot{\boldsymbol{S}} \dot{\boldsymbol{q}} + \boldsymbol{S} \ddot{\boldsymbol{q}} \tag{2.48}$$

Apart from projecting generalized velocities to spatial velocities, a joint model also determines how generalized coordinates $\boldsymbol{q}$ project to spatial transformations. The spatial transformation induced by the joint generalized coordinates is $\boldsymbol{X}_{J_i}(\boldsymbol{q})$ and is defined by the type of the joint. For example, a revolute joint on the X axis would define $\boldsymbol{X}_{J_i} = \mathrm{Xrot}_x(q)$. Lastly, the transformation that locates the origin of the joint in its parent frame is notated by $\boldsymbol{X}_{T_i}$, and is part of a specific model structure. The full transformation that locates the parent frame into the child frame is therefore given by

$$ {}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{J_i}\boldsymbol{X}_{T_i} \tag{2.49} $$

There are thus two entities that completely define any type of joint, its motion subspace, $\boldsymbol{S}_i$ and its joint transform $\boldsymbol{X}_{J_i}$.

Joint models in cȯdyn

cȯdyn does not model the physical bodies separately from the constraints imposed by joints connecting bodies together. Instead, it defines each physical body and its degrees of freedom, relative to its parent, in the same node. This approach is a result of the specific RBD algorithms used, in which this representation is natural. By doing so, a kinematic tree can be easily defined hierarchically with a minimal amount of nodes.

The general structure of a physical body, plus joint, in cȯdyn is provided in Model 2.16.

cȯdyn model 2.16 – General physical body and joint definition

```
templates {
    node "body" {
         q = 0
        dq = 0
         τ = 0

          m = 1
        com = "[0; 0; 0]"
          I = "eye(3)"

        spI = "Spatial.Inertia(com, m, I)"

        node "joint" {
            JointTransform(q)          = "eye(6)"
            JointVelocity(q, dq)       = "zeros(6, 1)"

            MotionSubspace(q, dq)      = "∂[JointVelocity; dq](q, dq)"
            BiasVelocityProduct(q, dq) = "∂[MotionSubspace; q](q, dq) ∗ dq^T ∗ dq"
        }

                         tr = "[0; 0; 0]"
        coordinateTransform = "Spatial.Translation(tr)"
                  transform = "JointTransform(q) ∗ coordinateTransform"

            velocity = "JointVelocity(q, dq)"
        acceleration = "transform ∗ [0; 0; 0; −g] + BiasVelocityProduct(q, dq)"

        baseToLocalTransform = "transform"
        localToBaseTransform = "Spatial.Inverse(baseToLocalTransform)"
    }
}
```

The quantities that are specific to a particular model and need to be defined are the mass `m`, center of mass `com`, inertia tensor `I`. Furthermore, the `coordinateTransform` ($\boldsymbol{X}_T$), locating the body origin in its parent frame, should be supplied. Finally, specific joint types need to define two functions, the `JointTransform` ($\boldsymbol{X}_{J_i}$) and the `JointVelocity` ($\boldsymbol{v}_J$). In practice, it is often easier to derive the `JointVelocity` function than to derive directly the `MotionSubspace`. Here we can see that the symbolic partial derivative functionality in cȯdγn can be used to automatically derive the correct joint `MotionSubpace` from a given `JointVelocity`, and also derives the `BiasVelocityProduct` automatically from the obtained `MotionSubspace`.

The `velocity` and `acceleration` define the body's spatial velocity and acceleration and are initially defined for the body having the fixed inertial frame as its parent (i.e. it is the root in the kinematic tree). In this case, the `velocity` of the body is simply its `JointVelocity` and the `acceleration` is the projected acceleration plus the bias velocity product. Note that acceleration due to gravity can be modeled as an external force as well, but it is easier to project it directly to the root acceleration of the system. This is exactly equivalent due to the fact that gravity can be seen as a constant linear acceleration.

Joint types

cȯdγn provides a variety of standard joint types in its standard library. Model 2.17 defines the most important ones, including revolute, prismatic, planar, spherical and floating joints. As can be seen, it is very easy to model different types of joints. Users can therefore also easily define their own joint types, directly using the cȯdγn language.

Two joint types are of particular interest, the `spherical` and `float` joints. Both are implemented using quaternions to represent the rotational degrees of freedom. Although it is more straightforward to represent the rotational degrees of freedom using Euler rotations (i.e. three variables, one for each rotation), singularities cannot be avoided in general. Quaternions do not suffer from this problem, however they do have their own. First of all, quaternion rotations are realized through the use of *unit* quaternions and it is important that they stay normalized. Due to numerical errors, the quaternion will tend to drift slowly over time. Fortunately, normalization can be easily achieved in cȯdγn through the use of *variable constraint expressions*. These expressions can be associated with any variable and provide an easy way to restrict variables to a certain domain, or in this case normalize them. The spherical and float joints in model 2.17 show the use of these *variable constraint expression* in their definition of q.

The other difficulty about representing rotations by quaternions is that quaternions require 4 (constraint) generalized coordinates, but only 3 generalized velocities, i.e. the number of generalized coordinates is not equal to the number of generalized velocities. Again, in cȯdγn this is not an issue, since differential equations can be specified separately. It is therefore only a matter of expressing the differential equation of the generalized coordinates $\boldsymbol{q}$ as the quaternion derivative of the generalized velocities

$$\dot{q} = \frac{1}{2}\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_3 & q_0 & q_1 \\ q_2 & q_1 & q_0 \end{bmatrix} \mathrm{d}q \tag{2.50}$$

ċodẏn model 2.17 – Various available joint types [play]

```
node "revolute{X,Y,Z}" : physics.body {
    node "joint" {
        axis = ["[1; 0; 0]", "[0; 1; 0]", "[0; 0; 1]"]

        JointVelocity(q, dq) = "[axis ∗ dq; 0; 0; 0]"
           JointTransform(q) = "Spatial.Rotation@@@1(q)"
    }
}

node "prismatic{X,Y,Z}" : physics.body {
    node "joint" {
        axis = ["[1; 0; 0]", "[0; 1; 0]", "[0; 0; 1]"]

        JointVelocity(q, dq) = "[0; 0; 0; axis ∗ dq]"
           JointTransform(q) = "Spatial.Translation(axis ∗ q)"
    }
}

node "planarY" : physics.body {
    node "joint" {
        JointVelocity(q, dq) = "[0; dq[0]; 0; 0; dq[1]; dq[2]]"
           JointTransform(q) = "Spatial.Translation([0; q[1]; q[2]) ∗ Spatial.
             RotationY(q[0])"
    }
}

node "spherical" : physics.body {
    node "joint" {
        JointVelocity(q, dq) = "[dq; 0; 0; 0]"
           JointTransform(q) = "Spatial.Quaternion(q)"
    }

    q = "[1; 0; 0; 0]" ("q / sqsum(q)")

    dqdot = "0.5 ∗ [−q[1], −q[2], −q[3];
                     q[3],  q[0], −q[1];
                    −q[2],  q[1],  q[0]] ∗ dq"
}

node "float" : physics.body {
    node "joint" {
        JointVelocity(q, dq) = "dq"
           JointTransform(q) = "Spatial.Translation(q[4:7]) ∗ Spatial.Quaternion(q
             [0:4])"
    }

    q = "[1; 0; 0; 0; 0; 0; 0]" ("[q[0:4] / sqsum(q[0:4]); q[4:7]]")

    dqdot = "[0.5 ∗ [−q[1], −q[2], −q[3];
                      q[3],  q[0], −q[1];
                     −q[2],  q[1],  q[0]] ∗ dq[0:3];
              dq[3:6] + Spatial.Cross(q[4:7], dq[0:3])"
}
```

Similarly, a special floating base joint allows for efficient modeling of a system with a floating base (e.g. a legged robot). It uses a quaternion representation for the rotation and 3 additional generalized coordinates for the translation. Floating bases are more commonly modeled by stacking massless revolute and prismatic joints, which can lead to instabilities and inefficiencies.

2.6.5 Model definition

Having defined joint models for various types of joints, we can now define rigid body models in ċdyn. Model 2.18 shows how a relatively simple N-pendulum model can be defined. First all bodies in the system are defined, in this case of joint type `revoluteY`. Then, the structure (or connectivity) of the system is defined by defining `physics.joint` edges between physical body nodes. As can be seen, the resulting model is not structured explicitly as a tree, which would result in deeply nested nodes. Instead, a flat structure is preferable, separating kinematic structure from model structure creating a comprehensible and succinct model representation.

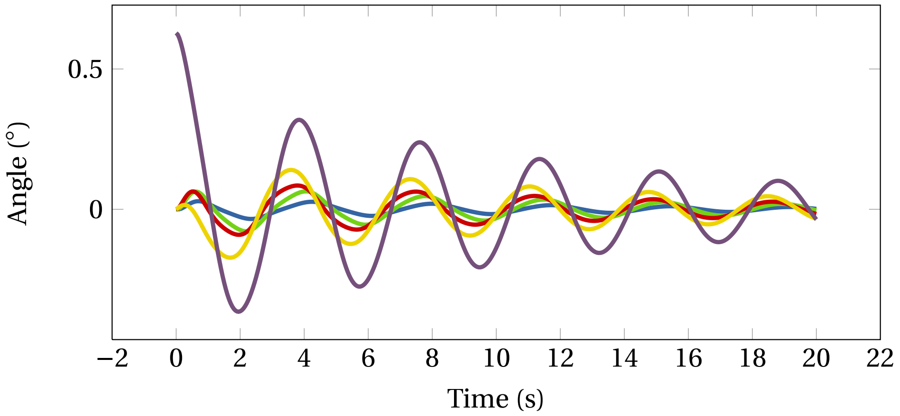


Figure 2.13 – System output of a rigid body dynamics simulation of a chain of 5 pendulums. The first pendulum starts out at an angle and starts accelerating due to gravity. The system is damped by a simple velocity based damping term on the generalized forces of each of the pendulum joints.

2.6.6 Inverse dynamics

Having previously defined all required quantities that define a RBD model, it is now time to turn to deriving the dynamics. We first start by deriving the inverse dynamics, i.e. determining $\boldsymbol{\tau}$ given $\ddot{\boldsymbol{q}}$, for the system. As we will see, we can actually use the same algorithm as part of deriving the forward dynamics.

The inverse dynamics can be obtained using a spatial version of the Recursive Newton-Euler Algorithm (RNEA). Given the quantities $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ and a given model specification (i.e. defining còdγn model 2.18 – Simple multi-pendulum model definition [play]

```
# Includes the codyn physics templates for systems, bodies and
# joint models
include "physics/physics.cdn"

integrator {
    method = "runge-kutta"
}

defines {
    n = 5
}

# All models start by defining a node derived from the physics.system
# template
node "system" : physics.system {
    # Inside the system, joints are defined by inheriting from any
    # of the physics.joints.* templates. codyn provides a large number
    # of general purpose joints.
    node "p{1:@n}" : physics.joints.revoluteY {
        # The center of mass
        com = "[0; 0; -0.5]"

        # The translation from the parent frame to the frame of
        # this joint
         tr = "[0; 0; -1]"

        # The inertia tensor of the physical body on this joint
          I = "Inertia.Box(m, 0.05, 0.05, 1)"

        # Add some damping in the system on the generalized force
        τ = "-20 * dq"
    }

    # Override certain variables on the root joint.
    node "p1" {
        tr = "[0; 0; 0]"
         q = "0.2 * pi"
    }

    # Create edges between successive nodes inheriting from
    # the physics.joint template. This chains joints together
    # to form the articulated rigid body.
    edge from "p{1:@n}" to "p$(@1 + 1)" : physics.joint {}

    # The physics/model.cdn file should be included at the end of the
    # model definition and constructs a "model" node containing global
    # system quantities such as the center of mass and total mass. It is
    # also responsible for constructing the required Jacobians if requested.
    include "physics/model.cdn"

    # The physics/dynamics.cdn file should be included last and constructs
    # the equations necessary for forward simulation of the dynamics. It
    # uses RNEA to construct C and CRBA to construct H in a new node called
    # "dynamics". It then derives generalized accelerations in dynamics.ddq
    # which project back to the individual joints.
    include "physics/dynamics.cdn"
}
```

$\boldsymbol{S}_i$, $\boldsymbol{v}_{J_i}$, $\boldsymbol{c}_{J_i}$ and ${}^i\boldsymbol{X}_{\lambda(i)}$), we can compute $\boldsymbol{\tau}$ recursively as follows:

$$\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{J_i} \tag{2.51}$$

$$\boldsymbol{a}_i = {}^i\boldsymbol{X}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\ddot{\boldsymbol{q}} + \boldsymbol{c}_{J_i}\boldsymbol{v}_i \times \boldsymbol{v}_{J_i} \tag{2.52}$$

$$\boldsymbol{f}_i = \boldsymbol{I}_i\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\boldsymbol{v}_i - {}^i\boldsymbol{X}_0^*\boldsymbol{f}_i^x + \sum_{j\in\mu(i)} {}^i\boldsymbol{X}_j^*\boldsymbol{f}_j \tag{2.53}$$

$$\boldsymbol{\tau}_i = \boldsymbol{S}^T\boldsymbol{f}_i \tag{2.54}$$

Here ${}^{i}\boldsymbol{X}_0$ is the spatial transform from base coordinates to i^{th} body coordinates and $\boldsymbol{f}_i^x$ are any external forces acting on body i, in base coordinates. $\mu(i)$ is the set of direct child bodies of i (i.e. $\lambda(j) = i$). Each of these bodies projects its force $\boldsymbol{f}_i$ to its parent through the spatial force transform ${}^{i}\boldsymbol{X}_j$.

Since cȯdγn is a declarative language, it is not possible to implement general algorithms. It is however possible to directly define all the necessary quantities directly from their mathematical form, and let cȯdγn resolve the recursive relationship automatically. Recursive Newton Euler can therefore be naturally described in cȯdγn, simply by directly translating the equations above. The procedure is as follows:

1. Add new variables `force`, `forceChild` and `forceExternal` to each physical body
2. Define `force` as per equation 2.53, using `forceChild` for forces projected from the children of the body
3. Use edge projection to project `velocity` and `acceleration` from each parent to each child
4. Use edge projection to project `force` from each child to their parents `forceChild`

2.6.7 Forward dynamics

If we look at the inverse dynamics using recursive Newton Euler, we can see that if we set $\ddot{\boldsymbol{q}}$ to 0, then we are computing $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ instead of $\boldsymbol{\tau}$. What we are left with is the computation of $\boldsymbol{H}(\boldsymbol{q})$. A reasonable algorithm to compute $\boldsymbol{H}(\boldsymbol{q})$ is called the Composite Rigid Body Algorithm (CRBA). A spatial version of this algorithm recursively projects spatial inertia into $\boldsymbol{H}$.

We start by computing the composite rigid body, spatial inertia using

$$\boldsymbol{I}_{c_i} = \boldsymbol{I}_i + \sum_{j \in \mu(i)} {}^{i}\boldsymbol{X}_j^{*} \boldsymbol{I}_{c_j} {}^{i}\boldsymbol{X}_j, \tag{2.55}$$

where $\boldsymbol{I}_{i_c}$ is the composite spatial inertia of body i in body i coordinates. It is simply the sum of all body inertias below body i in the kinematic tree, projected into the coordinate system of body i. Note the recursive definition which makes computation both efficient, as well as suitable for implementation in cȯdγn.

Having obtained $\boldsymbol{I}_{c_i}$, we can project the spatial inertia to $\boldsymbol{H}$ through the joint motion subspace as follows

$$\boldsymbol{H}_{ij} = \boldsymbol{S}_j^{T}\, {}^{j}\boldsymbol{F}_i \tag{2.56}$$

$$\boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^{T} \tag{2.57}$$

$${}^{i}\boldsymbol{F}_i = \boldsymbol{I}_{c_i} \boldsymbol{S}_i \tag{2.58}$$

$${}^{\lambda(j)}\boldsymbol{F}_i = {}^{\lambda(j)}\boldsymbol{X}_j^{*}\, {}^{j}\boldsymbol{F}_i, \tag{2.59}$$

equation 2.59 is of particular interest. It recursively computes the contribution of the composite spatial inertia from each body i, upwards to each parent j in the tree.

The procedure in cȯdγn to implement these equations is as follows:

1. Add new variables `ICChild` and `IC` for every body, where
   `IC = "spI + ICChild"`
2. Declare edge projections to project `IC` from every child to its parent through
   `ICChild += "transform`^{T}`* IC * transform"`
3. Add new variables
   `iFi = "IC * MotionSubspace(q, dq)"`
   `Hii = "MotionSubspace(q, dq)`^{T}`*`$^i\boldsymbol{F}_i$`"`
4. For every body i, declare edge projections to project to each body j, on the path from i to the base
   `jFi <= "transform`^{T}`* iFi"`
5. For every body i, declare edge projections for every body j, on the path from i to the base, towards $\boldsymbol{H}$
   `Hij <= "MotionSubspace(q, dq)`^{T}`* jFi"`
   `Hji <= "Hij`^{T}`"`

Both the inverse and forward dynamics are implemented *completely* using only the declarative language. This is important because the result is that we end up with a fully declarative model of the derivation of the equations of motion. It is simply defined directly in terms of physical quantities, leading to a very natural and efficient representation.

2.6.8 Jacobian

The Jacobian is an important quantity in rigid body dynamics, modeling, robotics and control which we will need in chapter 5 for the modeling of wearable parallel structures. In rigid body dynamics, the term Jacobian refers to a mapping between generalized velocities and Cartesian velocities (in some frame). As such, it is the partial derivative of a Cartesian point $\boldsymbol{x}$ towards generalized coordinates, i.e.

$$\boldsymbol{J}(\boldsymbol{x}(\boldsymbol{q})) = \begin{bmatrix} \dfrac{\partial x_1}{\partial q_1} & \cdots & \dfrac{\partial x_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial x_m}{\partial q_1} & \cdots & \dfrac{\partial x_m}{\partial q_n} \end{bmatrix} \tag{2.60}$$

Furthermore, recall that the total time derivative of $x_i(\boldsymbol{q})$ is given by

$$\dot{x}_i = \sum_j^n \frac{\partial x_i}{\partial q_j} \dot{q}_j \tag{2.61}$$

or

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}} \tag{2.62}$$

As mentioned before, it is actually not needed to compute the Jacobian by taking an actual partial derivative. In fact, equation 2.62 should look familiar, we have already seen it before when defining how joints transform a generalized velocity to a spatial velocity

$$\boldsymbol{v}_i = \boldsymbol{S}_i\dot{\boldsymbol{q}}_i \tag{2.63}$$

Since spatial velocities can be summed (as long as they are expressed in the same frame), we can obtain a base Jacobian ${}^0\boldsymbol{J}$, mapping generalized velocities to base velocities, as follows

$${}^0\boldsymbol{J} = \begin{bmatrix} {}^0\boldsymbol{X}_0\boldsymbol{S}_0 & \cdots & {}^0\boldsymbol{X}_n\boldsymbol{S}_n \end{bmatrix} \tag{2.64}$$

We can obtain a Jacobian from a particular body i to the base, ${}^0\boldsymbol{J}_i$ simply by selecting the columns from ${}^0\boldsymbol{J}$ corresponding to the generalized velocities of the bodies on the kinematic path from body i to the base

$${}^0\boldsymbol{J}_i = \begin{bmatrix} {}^i\epsilon_0{}^0\boldsymbol{J}_0 & \cdots & {}^i\epsilon_n{}^0\boldsymbol{J}_n, \end{bmatrix} \tag{2.65}$$

where ${}^i\epsilon_j$ is 1 if body j is on the path from i to the base, or 0 otherwise. ${}^0\boldsymbol{J}_i$ is a mapping from generalized velocities to Cartesian velocities at the system origin. However, often one will want to obtain the Jacobian relating velocities to those observed at a certain end-effector position.

Because the Jacobian is a mapping from generalized velocities to spatial velocities, we can easily transform it to a different coordinate frame simply by applying a spatial motion transform. Therefore, to obtain the Jacobian at a certain end effector position ${}^0\boldsymbol{r}$ (in base coordinates), we can apply a spatial motion translation:

$${}^0\boldsymbol{J}_{i_r} = \texttt{Xtr}({}^0\boldsymbol{r}){}^0\boldsymbol{J}_i \tag{2.66}$$

We can also easily obtain the Jacobian for a particular body in a different base. This type of Jacobian describes the mapping of generalized velocities to relative velocities between two bodies and is very useful in kinematics based control (for example, controlling the end of one leg relatively to the end of another leg). Given the base Jacobian ${}^0\boldsymbol{J}_{j_r}$ for a body j at a position ${}^0\boldsymbol{r}_j$ and similarly the base Jacobian ${}^0\boldsymbol{J}_{i_r}$ for a body i at a position ${}^0\boldsymbol{r}_i$, the Jacobian for j at ${}^0\boldsymbol{r}_j$ assuming the base i at ${}^0\boldsymbol{r}_i$ is given by:

$${}^i\boldsymbol{J}_j = {}^0\boldsymbol{J}_{j_r} - \texttt{Xtr}({}^0\boldsymbol{r}_j - {}^0\boldsymbol{r}_i){}^0\boldsymbol{J}_{i_r} \tag{2.67}$$

Finally, another important Jacobian to derive is the center of mass Jacobian. This Jacobian relates generalized velocities to center of mass velocities, and is another important quantity for control (in particular for kinematics based balance control). Having calculated the center of mass CoM by

$$\mathbf{CoM} = \frac{1}{M}\sum_i \left(m_i \,\texttt{XtrP}\,({}^0\boldsymbol{X}_i, \mathbf{CoM}_i)\right) \tag{2.68}$$

$$M = \sum_i m_i \tag{2.69}$$

we can obtain the center of mass Jacobian ${}^{\text{CoM}}\boldsymbol{J}$ in a similar way as the base Jacobian:

$${}^{\text{CoM}}\boldsymbol{J} = \begin{bmatrix} {}^0\boldsymbol{X}_0\,\texttt{Xtr}\,(\mathbf{CoM})\boldsymbol{S}_0 & \cdots & {}^0\boldsymbol{X}_n\,\texttt{Xtr}\,(\mathbf{CoM})\boldsymbol{S}_n \end{bmatrix} \tag{2.70}$$

In ċdγn, all these Jacobians are readily available. A Jacobian node can be defined inside any physical body i, at a location $\boldsymbol{r}_i$ inside the local frame of that body, resulting in the computation of ${}^0\boldsymbol{J}_{\boldsymbol{r}_i}$. It does so by first computing ${}^0\boldsymbol{J}$, the base Jacobian of the full system and then projecting back the relevant subsections of ${}^0\boldsymbol{J}$ to each required body Jacobian. The general procedure is as follows:

1. Determine if any Jacobians need to be computed (i.e. are there any nodes with the `physics.jacobian` template)
2. If so, create a new variable `J0` in the system `jacobian` node, containing the base Jacobian ${}^0\boldsymbol{J}$ computed by edge projecting columns of ${}^0\boldsymbol{J}$ from each body in the system
3. Then, project back relevant columns of ${}^0\boldsymbol{J}$ (i.e. those in the path from body i to the base, the joints which contribute to velocities in i) to each `physics.jacobian` node

Model 2.19 shows a basic usage of Jacobians in a ċdγn model. The center of mass Jacobian is also automatically available in all systems and can be obtained from `jacobian.Jcom`. Finally, Jacobians can easily assume a different base using equation 2.67, made available conveniently in a ċdγn file which can be included inside a physical body node.

2.6.9 Closed loop dynamics

Whenever a kinematic structure closes on itself, the dynamics of that structure change significantly. The *mobility* is defined as the degree of motion allowed by a certain structure. For a non-closed kinematic tree, this is simply equal to the number of degrees of freedom. However, whenever a structure closes on itself, it removes n_c degrees of freedom, where n_c depends on the type of closing joint constraint. For example, consider a planar N-dof series manipulator consisting of N revolute joints. If we close this structure from end point to base with another revolute joint, we obtain a new system of only $N-3$ degrees of motion. Modeling of closed loop dynamics is essential for the co-design of wearable robots presented in chapter 5.

To obtain the equations of motion for closed loop systems, there are generally two methods cȯdγn model 2.19 – Example usage of Jacobians [play]

```
node "system" : physics.system {
    node "p{1:4}" : physics.joints.pendulumY {
        tr = "[0; 0; -1]"
    }

    node "p4" {
        node "jac" : physics.jacobian {
            location = "[0; 0; -1]"
        }

        # Construct a jacobian for the center of mass Jacobian,
        # assuming the base in the p4 phsyical body. The resulting
        # jacobian is available in JComInP4 and maps generalized
        # velocities to velocities of the center of mass as observed
        # from p4.jac.location
        parse "physics/algorithms/rebase_jacobian.cdn" {
            tipJacobian = "jacobian.Jcom"
            tipLocation = "model.com"
                   base = "jac"
                    var = "JComInP4"
        }
    }

    edge from "p{1:4}" to "p$(@1 + 1)" : physics.joint {}

    # Processes all requested jacobians and creates corresponding
    # edge projections to obtain (in this case) p4.jac.J, the body
    # Jacobian of p4 at p4.jac.location
    include "physics/model.cdn"
}
```

(Featherstone, 2008). The first method is to transform the system to obtain a new, reduced set of generalized coordinates $\boldsymbol{y}$. Given that $\boldsymbol{y}$ defines $\boldsymbol{q}$ uniquely, a function $\boldsymbol{\gamma}$ exists such that $\boldsymbol{q} = \boldsymbol{\gamma}(\boldsymbol{y})$. I.e. $\boldsymbol{\gamma}$ is the function that maps the reduced generalized coordinates $\boldsymbol{y}$ to the original generalized coordinates $\boldsymbol{q}$. Then, derive twice to obtain the same mapping for generalized velocities and accelerations

$$\dot{\boldsymbol{q}} = \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{y}} \dot{\boldsymbol{y}} \tag{2.71}$$

$$\ddot{\boldsymbol{q}} = \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{y}} \ddot{\boldsymbol{y}} + \left(\frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{y}} \frac{\mathrm{d}}{\mathrm{d}t} \right) \dot{\boldsymbol{y}} \tag{2.72}$$

The advantage of this method is that no explicit constraints have to be introduced into the equations of motion, resulting in stable and accurate simulations. The main difficulty of this approach is that it is not always possible to find a mapping function $\boldsymbol{\gamma}$ which uniquely maps $\boldsymbol{y}$ to $\boldsymbol{q}$. More so, it is especially difficult to obtain $\boldsymbol{\gamma}$ automatically for general structures. Finally, closed loop systems can loose degrees of freedom, for example in singular configurations. When this happens, it might be necessary to change the set of independent coordinates $\boldsymbol{y}$ such that the system is no longer singular in the choice of $\boldsymbol{y}$. This would have to happen during simulation and is an expensive, and complicated operation.

The second method is more general and easier to use since it derives the equations of motion for any closed system automatically. It does so by introducing acceleration constraints in the original equations of motion. In the original equations of motion, there will be two new generalized forces 1) τ_c, the *unknown* constraint forces which account for the kinematic constraint imposed by the kinematic loop and 2) τ_a the *known* active forces exerted on the loop joint (actuator forces, springs, dampers etc.). We can define the kinematic constraint imposed by the loop joints in terms of joint acceleration by

$$\boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k} \tag{2.73}$$

the same constraint imposes loop constraint forces by

$$\boldsymbol{\tau}_c = \boldsymbol{K}^T \boldsymbol{\lambda} \tag{2.74}$$

and the new equations of motion will become

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{K}^T \\ \boldsymbol{K} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}_{\boldsymbol{a}} \\ \boldsymbol{k} \end{bmatrix} \tag{2.75}$$

The objective is then to construct $\boldsymbol{K}$ and $\boldsymbol{k}$, solve for the unknown Lagrange multiplier $\boldsymbol{\lambda}$ and finally solve for the accelerations $\ddot{\boldsymbol{q}}$.

The motion that a loop constraint joint allows can be defined by a matrix $\boldsymbol{T}$ called the constraint force subspace. It is the orthogonal complement of the joint motion subspace, i.e.

$$\boldsymbol{T} = \boldsymbol{S}^{\perp} \tag{2.76}$$

and

$$\boldsymbol{T}^T \boldsymbol{S} = 0 \tag{2.77}$$

in other words, $\boldsymbol{T}$ projects generalized forces into the constraint force subspace. Note that $\boldsymbol{T}$ can be any basis that spans the correct subspace and there is thus no unique choice of $\boldsymbol{T}$. An example of a possible choice for $\boldsymbol{T}$ for a revolute joint on the X axis (and corresponding motion subspace $\boldsymbol{S}$) is given in equation 2.78.

$$\boldsymbol{S} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.78}$$

Having defined $\boldsymbol{T}$, we can see that a constraint imposed using $\boldsymbol{T}$ leads to

$$\boldsymbol{T}^T \boldsymbol{v}_J = \boldsymbol{0} \tag{2.79}$$

i.e. the velocity across the loop joint, projected through the constraint force subspace must be $\boldsymbol{0}$ (allow no motion). The loop joint velocity $\boldsymbol{v}_J$ is simply the difference between the two spatial velocities of the bodies the loop closes over. To obtain acceleration constraints, equation 2.79 can be differentiated, which after simplification (see Featherstone (2008) for details) results in:

$$\boldsymbol{T}_{ij}^T(\boldsymbol{J}_i - \boldsymbol{J}_j)\ddot{\boldsymbol{q}} = -\boldsymbol{T}_{ij}^T(\boldsymbol{a}_i^{\text{vp}} - \boldsymbol{a}_j^{\text{vp}}) - \dot{\boldsymbol{T}}_{ij}^T(\boldsymbol{v}_i - \boldsymbol{v}_j) \tag{2.80}$$

Here $\boldsymbol{a}_i^{\text{vp}}$ is the *velocity product* acceleration of joint i and is available as a by-product from applying recursive Newton Euler. The body Jacobians $\boldsymbol{J}_i$ and $\boldsymbol{J}_j$ do not need to be explicitly derived, and can be obtained by projecting the motion subspaces of the two joints in the correct frame of reference as explained in the previous section. We have thus obtained $\boldsymbol{K}$ and $\boldsymbol{k}$ for the closed loop system.

To solve for $\boldsymbol{\lambda}$ we can write 2.75 equivalently as

$$\boldsymbol{A\lambda} = \boldsymbol{b} \tag{2.81}$$

$$\boldsymbol{A} = \boldsymbol{K}\boldsymbol{H}^{-1}\boldsymbol{K}^T \tag{2.82}$$

$$\boldsymbol{b} = \boldsymbol{k} - \boldsymbol{K}\boldsymbol{H}^{-1}(\boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}_{\boldsymbol{a}}) \tag{2.83}$$

and solve for $\boldsymbol{\lambda}$. Note that $\boldsymbol{A}$ is not generally invertible (i.e. $\boldsymbol{\lambda}$ does not have a unique solution), but the equation can still be solved (for example using the pseudo inverse). We can also avoid the explicit computation of $\boldsymbol{H}^{-1}$. Based on the method proposed in (Featherstone, 2008), we use an $\boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$ decomposition of $\boldsymbol{H}$ to write

$$\boldsymbol{H} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L} \tag{2.84}$$

$$\begin{aligned}\boldsymbol{H}^{-1} &= (\boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L})^{-1} \\ &= \boldsymbol{L}^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-T}\end{aligned} \tag{2.85}$$

$$\begin{aligned}\boldsymbol{A} &= \boldsymbol{K}\boldsymbol{L}^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-T}\boldsymbol{K}^T \\ &= \boldsymbol{Y}^T\boldsymbol{D}^{-1}\boldsymbol{Y}\end{aligned} \tag{2.86}$$

$$\boldsymbol{Y} = \boldsymbol{L}^{-1}\boldsymbol{K}^T \tag{2.87}$$

$$\boldsymbol{b} = \boldsymbol{k} - \boldsymbol{Y}^T\boldsymbol{D}^{-1}\boldsymbol{L}^{-T}(\boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}_{\boldsymbol{a}}) \tag{2.88}$$

$$\boldsymbol{\lambda} = \boldsymbol{A}^{+}\boldsymbol{b} \tag{2.89}$$

Although this looks more complicated, the computation of $\boldsymbol{L}^{-1}\boldsymbol{x}$, $\boldsymbol{D}^{-1}\boldsymbol{x}$ and $\boldsymbol{D}^{-1}\boldsymbol{L}^{-T}\boldsymbol{x}$ can be performed efficiently using sparse methods adopted from (Featherstone, 2008).

Finally, having obtained $\boldsymbol{\lambda}$, we can solve for $\ddot{\boldsymbol{q}}$ in

$$\boldsymbol{H}\ddot{\boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}_a + \boldsymbol{K}^T\boldsymbol{\lambda} \tag{2.90}$$

The main disadvantage of this method is that due to numerical and integration errors, the kinematic constraints tend to drift. To address this issue, a stabilizing force can be added to the computation of $\boldsymbol{k}$ which tries to reduce errors appearing in this way. The resulting system is less accurate and less stable, so care has to be taken when analyzing results obtained from simulations in this way.

In cȯdγn, loop joints are defined by loop joint nodes. Several templates are provided for different types of loop closing joints, specifying their corresponding constraint force matrices. Loop joint edges then specify how a loop joint connects between two physical bodies in the system.

The algorithms as outlined above are implemented in cȯdγn by overriding the computation of the forward dynamics in case of loop closing joints. If any loop closing joints are found, the additional required equations are automatically added based on the closing loop joint types.

Example 1: loop closure to fixed base

The first example is a model of a simple parallel structure which closes on the fixed base. Model 2.20 shows the cȯdγn model for this example. It defines 3 revolute joints and then closes the last body on the fixed world using a `physics.cjoints.revoluteY` closing joint. As can be seen in the model, creating a closed loop system is very simple and does not require other modifications to the original, non closed system.

Figure 2.14 shows the output and a schematic representation of the system. Note that the original system has 3 degrees of freedom, but the resulting closed system has only one. This can be easily seen in the output of the system since the motions can be described by one variable (oscillations at the same amplitude, phase and frequency). A video of the resulting simulation can be found on the cȯdγn website.

Example 2: Pantographic leg

While the previous example was closed on the fixed base, the next example models a parallel structure of a pantographic leg (or 4 bar parallel mechanism). The schematic representation of such a structure is shown in figure 2.15 on the left. The original system has 4 degrees of freedom. After adding the closing loop constraint, the system reduces to 2 degrees of freedom.

cȯdẏn model 2.20 – Example of a loop closing joint [play]

```
include "physics/physics.cdn"
include "physics/cjoints.cdn"

integrator {
    method = "runge−kutta"
}

defines {
    offset = "0.2 ∗ pi"
}

node "system" : physics.system {
    node "p{1:3}" : physics.joints.revoluteY {
         tr = "[0; 0; −1]"
        com = "[0; 0; −0.5]"
          I = "Inertia.Box(m, 0.05, 0.05, 1)"
    }

    node "p1" {
        tr = "[0; 0; 0]"
         q = "−@offset"
    }

    node "p{2:3}" {
        q = ["0.5 ∗ pi + @offset", "0.5 ∗ pi − @offset"]
    }

    edge from "p{1:3}" to "p$(@1 + 1)" : physics.joint {}

    # Create a closing joint node
    node "pcl" : physics.cjoints.revoluteY {
        tr = "[0; 0; −1]"
    }

    # Connect the last body (p3) to the closing loop joint
    edge from "p3" to "pcl" : physics.cjoint {}

    include "physics/model.cdn"
    include "physics/dynamics.cdn"
}
```

The model is provided in model 2.21. It is similar to the previous closed model, except that the closing joint, `cl`, now closes back on the `hip` instead of the fixed base. In the model, a small damping force is added on the hip and a small spring force is added on the knee joint. The output of this system when simulated is shown in figure 2.15 on the right. The knee stiffness causes high frequent oscillations in the knee joint due to initial movement of the hip (due to gravity). The hip joint shows a damped oscillatory movement until the system comes to a rest. A video of the resulting simulation can be found on the cȯdẏn website.

2.6.10 Contact modeling

The contact model is an essential component of a general purpose simulator, in particular if used for the study of locomotion. Contact dynamics is a hard problem and has a whole field

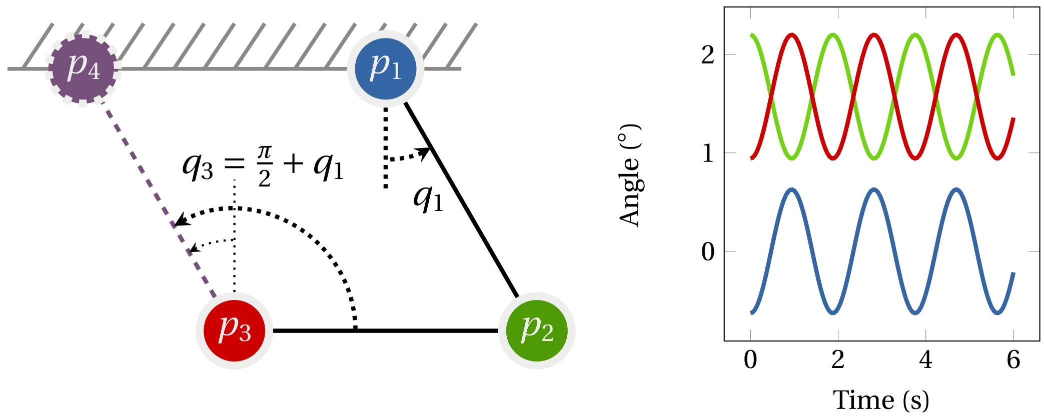


Figure 2.14 – Left: a schematic representation of a closed loop system of one degree. The original system consists of 3 revolute joints and is closed to the fixed base through p_4. Right: Output of the simulated system. The resulting joint angle motions are consistent with the imposed constraint and can be described by a single variable.

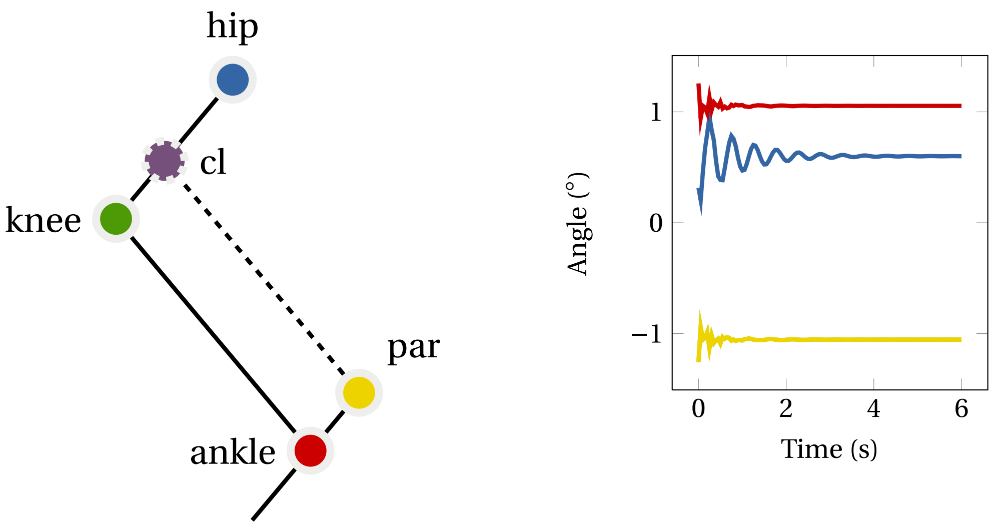


Figure 2.15 – Left: a schematic representation of the leg. The open loop system consists of 4 revolute joints, hip, knee, ankle and par. A closing joint, cl, creates a pantographic structure having only 2 degrees of freedom. Right: output of simulating the system. A damping force acts on the hip and a spring force acts on the knee, causing quick oscillations.

of research dedicated to it. Here we only briefly discuss the two most common methods and their implementation in cȯdyn.

For rigid body dynamics, there are generally speaking two approaches to modeling contacts (Featherstone, 2008), both of which are implemented in cȯdyn. The first, and reasonably straightforward, way is to model contact points by means of stiff spring/damper systems, and is referred to as soft contact modeling, or the penalty method for contact modeling. Basically, whenever there is a penetration between two surfaces, a virtual spring and damper exerts force on the penetrating bodies in the direction of their normal and tangential directions.

cȯdẏn provides a basic implementation of soft contacts with basic Coulomb friction. It uses cȯdẏn's event system to determine the timing of contact activation and deactivation, as well as to determine when a contact switches between stick and slip (depending on the friction coefficient). It does so by allowing the insertion of contact points in physical bodies. Whenever a contact point penetrates a contacting surface, contact forces are exerted as external forces on the corresponding physical bodies. The procedure is generally as follows:

1. Each contact can be in one of three states, *inactive*, *active* or *slipping*.
2. A contact point transitions from *inactive* to *active* whenever it penetrates a contacting surface
3. In the *active* state, a surface normal contact force is applied as the external force of the body in which the contact is embedded. This force is computed by

   $$\boldsymbol{f} = K\Delta p - D\dot{p} \tag{2.91}$$

   , with K the contact normal stiffness, Δp the penetration depth, D the contact normal damping and $\dot{p}$ the penetration velocity. The tangential (friction) force is computed in the same way, but tangentially to the contact surface.
4. The *active* state transitions to the *slipping* state when the tangential friction force exceeds the normal force times the friction coefficient.
5. When either in *active* or *slipping* state, the contact point can transition to the *inactive* state when contact between the two is broken (i.e. no longer penetrating).

The name *penalty method* refers to the fact that a contact reaction force only appears when there is a non-zero penetration, thus only when the contact constraint has already been violated. To make the system behave reasonably, the contact model needs to be very stiff, leading to issues in numerical integration. Soft contact models are relatively easy to implement, but very hard to tune. The spring and damper constants need to be finely tuned on a per model basis, depend on the number of contacts and even depend on the type of desired dynamic behavior of the system (for example walking and running might require different contact model constants). Soft contact models can often be the cause of instabilities, non-physical behavior and inaccuracies during simulation, in particular when using non-adaptive numerical integration methods. This method is used in section 4.1 of chapter 4.

The second method for contact modeling is called hard contact modeling. Hard contact models work by considering contacts as inequality constraints in the equations of motion. This leads to a hybrid definition of the equations of motion, such that the dynamics of the system are no longer defined by a single model. Effectively, hard contact models switch between different dynamical models of the system and any combination of active and inactive contacts leads to new dynamics. They are considerably harder to implement and are computationally much more expensive, but at the same time more accurate and more stable than soft contact models. In addition to switching between different dynamic models, impulse dynamics have to be propagated through the system at the time of contact, before switching to the new dynamics. This method is used in section 4.2 of chapter 4 and in chapter 5.

còdүn provides a basic hard contact model by utilizing the event system for opening and closing contact resolution and the closed loop dynamics developed in the previous section to implement the equality constraints imposed by active contacts.

Contact activation, impulse dynamics

When a contact point becomes active, an impulse has to be propagated through the system, such that the point of contact is at rest. Considering $\boldsymbol{r}$ the contact point, $\dot{\boldsymbol{r}}$ the contact point velocity and $\boldsymbol{J_r}$ the contact Jacobian, we require that the system after impulse is consistent with the following constraint:

$$\dot{\boldsymbol{r}} = \boldsymbol{J_r}\dot{\boldsymbol{q}}^{+} = 0 \tag{2.92}$$

i.e. we require a new state for the joint velocities, $\dot{\boldsymbol{q}}^{+}$, such that the contact point velocity $\dot{\boldsymbol{r}}$ is zero. To calculate the effect of the impulse on the joint velocities, we can use the inertia-weighted pseudo inverse of the contact Jacobian, also known as the dynamically-consistent generalized inverse of the Jacobian (Khatib, 1987).

$$\bar{\boldsymbol{J}}_{\boldsymbol{r}} = \boldsymbol{H}^{-1}\boldsymbol{J}_{\boldsymbol{r}}^{T}\boldsymbol{\Lambda}_{\boldsymbol{r}} \tag{2.93}$$

$$\boldsymbol{\Lambda}_{\boldsymbol{r}} = (\boldsymbol{J}_{\boldsymbol{r}}\boldsymbol{H}^{-1}\boldsymbol{J}_{\boldsymbol{r}}^{T})^{-1} \tag{2.94}$$

and we have

$$\dot{\boldsymbol{q}}^{+} = \bar{\boldsymbol{J}}_{\boldsymbol{r}}\dot{\boldsymbol{r}} \tag{2.95}$$

Note that this is the unique solution which minimizes $\dot{\boldsymbol{q}}^{T}\boldsymbol{H}\dot{\boldsymbol{q}}$. In còdүn contact activation/deactivation is modeled using events. This means that there is no global resolution of all contacts which are activated and deactivated. Rather, contacts are handled locally and sequentially. To account for constraints imposed by currently active contacts, we use an extended Jacobian, ${}^{+}\boldsymbol{J_r}$ which incorporates the active contact constraints:

$${}^{+}\boldsymbol{J_r} = \begin{bmatrix} \boldsymbol{J_r} \\ \boldsymbol{K} \end{bmatrix} \tag{2.96}$$

Doing so ensures the propagation of the impulse dynamics is consistent with the constraints imposed by the active contacts.

We thus obtain a new velocity state of the system. This causes a discrete change in generalized velocities as a result of the contact becoming active. In cȯdγn we compute this new state each time when a hard contact becomes active, i.e. when the event fires, and update new joint velocities accordingly.

Loss of contact

Whenever the constraint force which keeps the contact constraint active becomes negative, the contact is made inactive and there is a loss of contact. To obtain the contact force, recall from equation 2.81 that we solve for the Lagrange multipliers, $\boldsymbol{\lambda}$. In this case, the $\boldsymbol{\lambda}$ solved for represent the physical force required to keep the constraint. Due to the formulation of $\boldsymbol{K}$, $\boldsymbol{T\lambda}$ represents the constraint force in the body frame. We can then use a spatial force transformation to obtain the contact force in the frame of the contacting surface normal.

cȯdγn does not solve the linear complementary problem (LCP) when a contact becomes inactive, instead relying on sequential resolution of contact activation and deactivation from the event system. Although this is generally speaking inaccurate and can lead to jitter between contact points, we empirically found the approximation reasonable for systems with a small number of contacts, such as humanoid or quadrupedal models.

2.6.11 Visualization

cȯdγn does not provide any visualization of rigid body models by itself. Nevertheless, it is often very useful to be able to visualize the resulting 3D structure of a model. Problems in a model can be easier to find by simple visual inspection, and visualizing a resulting movement, instead of looking at joint angles, gives a much more intuitive representation of the system.

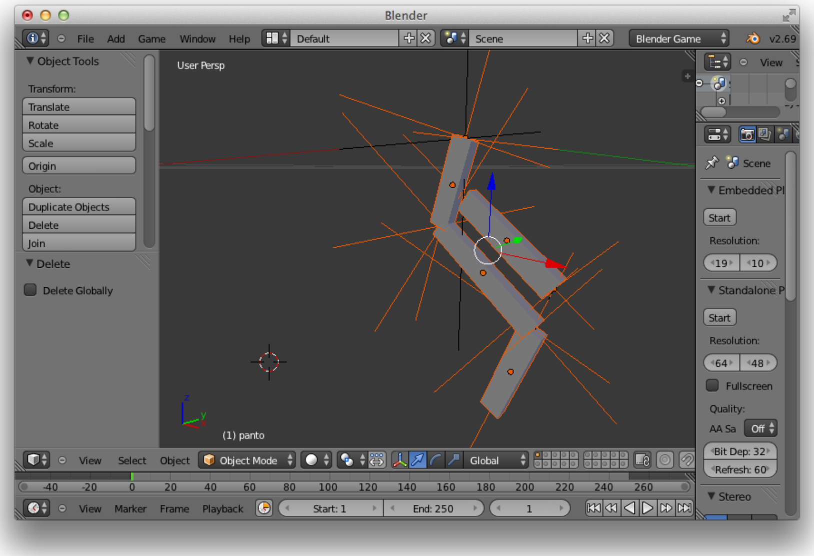
To aid in the design and inspection of rigid body models, cȯdγn integrates with the open source 3D studio, Blender. cȯdγn rigid body models can be imported directly from the cȯdγn model file. Furthermore, cȯdγn integrates with the Blender game engine to provide visualized forward simulation of the system at interactive rates.

2.7 Performance

It has been stressed a few times that cȯdγn aims for high performance and that it has a good conceptual model to this end. In this section we will see how it achieves this goal.

2.7.1 libcodyn

Until now, nothing has been said about the implementation of the execution engine in cȯdγn. The core library, *libcodyn*, is responsible for model parsing and evaluation. It does so by implementing a special purpose, stack based Virtual Machine specifically targeted for numerical computation. Mathematical expressions are compiled to low level instructions which are



Resolution: 19 10

Resolution: 64 48

Quality:

AA Sa Off

Bit Dep: 32

Refresh: 60

Start: 1

End: 250

1

Figure 2.16 – Screenshot of cȯdyn integration in blender. It shows a rendering of the closed loop system presented in model 2.21. Models can be directly imported from cȯdyn files and simulated using the Blender game engine.

then interpreted during execution. There are a few reasons that make the implementation relatively simple. Firstly, the only values cȯdyn math knows or cares about are floating point numbers and the stack is therefore simply a continuous array of floating point numbers. Secondly, cȯdyn has been designed with a dimension invariable runtime. This means that all instructions, expressions, variables and functions have a known and invariable size during execution. This in turn means that stack sizes are known beforehand and all required memory can be preallocated.

Although the libcodyn implementation is certainly not slow, it is also definitely not yet high performance. This becomes apparent especially when simulating large systems. Not only does it need to execute a virtual language, instead of native instructions, it also:

- Copies more memory than required due to each expression having its own stack
- Cannot use special purpose vectorization instructions (i.e. SIMD)
- Cannot optimize loops or memory access
- Cannot inline function calls
- Executes any and all instructions, even if not necessarily required

There is one advantage of the Virtual Machine implementation though. Because it does not do any optimizations, it can begin execution directly, without any overhead from a compilation stage. It is therefore perfectly suitable for quick experimentation, even for larger systems. Of course, there is no particular reason why libcodyn could not implement a more sophisticated execution engine, but implementing one is far from a trivial task, especially without loss of expressiveness in the language.

To compare the performance, we did a simple experiment simulating a coupled network of 20 amplitude controlled phase oscillators. We compare the performance of matlab (using a vectorized implementation) and standard libcodyn (using an implementation with nodes and edges, i.e. devectorized). Both in Matlab and libcodyn we use a Runge Kutta integrator of order 4 and simulate the differential equations for 10 seconds with a timestep of 1 millisecond. Simulating this system in Matlab (2014a) takes approximately 3.9 seconds while libcodyn (3.6) simulates it in approximately 3.3 seconds.

This shows that libcodyn is sufficient to simulate these type of differential equations with a performance very similar to that of Matlab. Furthermore, libcodyn can simulate up to 50 of such oscillators in real time (with full coupling) on a very moderate system, which makes it suitable for a large variety of applications.

Although this performance is sufficient for oscillator systems (which are arguably relatively simple computationally), this is not the case for larger and more complex systems such as the rigid body dynamics.

2.7.2 The road towards performance

There were two main motivations to address the issue of performance. The first and obvious motivation was simply that the rigid body dynamics simulations were not competitive in terms of simulation speed. This is especially problematic when using evolutionary optimization strategies or other population based optimization methods. These rely on large numbers of simulations to be performed and a simulator which is 10 times slower means 10 times fewer experiments.

Secondly, cȯdγn has a specific goal to run on embedded systems, such as hard real time constrained systems, but also micro-controllers and other micro systems with a very limited amount of resources (but capable of floating point arithmetic). Clearly, it is not possible to run any kind of high level virtual machine or libraries designed to run on consumer hardware on such platforms.

To deliver on the claims made, the cȯdγn framework provides a sophisticated tool which translates, without loss of functionality nor conceptual compromises, high level cȯdγn models into a representation which is:

- Low level
- Dependency free
- Real time ready
- Micro-controller compatible

2.7.3 As *raw* as *C*

The goal is to generate code that can run on low level or embedded systems with efficiency close to that of hand written code. It is possible to implement an actual compiler to translate directly into native instructions, but the amount of work involved in doing so would be a gigantic task and would likely not result in anything near the performance of what existing compilers produce.

Instead, ċodẏn models are translated into *C*, which is the most widely supported and low-level language currently available. Since this should result in *raw* performance, the tool has been aptly named *rawc*. Code generation is split into 5 stages:

1. Model destructuring
2. Sparsity analysis
3. Structure recovery
4. Dependency resolution
5. Abstract program generation
6. Code generation

Stage 1: Model destructuring

The first stage concerns reading in the model and performing analysis on the structure of the model. Because we are concerned about a low level implementation of the model, we can actually throw away all of the structure imposed by the ċodẏn language such as the nodes and edges. These concepts are required for the modeling methodology, but not for the actual generated code.

We therefore start by extracting all expressions which necessarily need to be computed. These include state variables, random values, differential equations and any dependencies of these (such as other variables, user functions, etc.). We can do so by first compiling the model to the virtual machine representation using libcodyn and then extracting all the generated virtual instructions.

Data storage in the low-level generated code is a single big block of floating point numbers called the `DataTable`. It contains storage for state variables, derivatives, delays, random numbers and any other intermediate values. Due to the fact that ċodẏn is a declarative language, all computable expressions are in the form of value assignments. The output of the analysis stage is then simply a list of `DataTable` assignments with pre-compiled virtual instructions.

Stage 2: Sparsity analysis

A very important step in generating efficient code is the sparsity analysis stage. Looking at the rigid body dynamics, it is quite obvious that by writing all the equations using spatial vector algebra, we cannot obtain efficient code directly. Quantities such as the motion subspace, force constraint subspace, spatial transformations and inertias are *all* sparse. The objective is to automatically determine the sparsity of these quantities and generate code that only computes values as necessary.

Because cȯdγn models are declared, the full model sparsity can be easily determined by iterating over all expressions, accumulating sparsity information over mathematical operators and functions, variable references, matrix and index operations, etc. It is important to note that this would be generally impossible in general purpose languages, unless the user would specifically indicate which expressions would be sparse.

Once the sparsity of all expressions is determined, a heuristic algorithm determines whether or not it is worth to generate special purpose sparse code for a particular operation, based on the number of sparse entries and number of reduced computations. If needed, instructions are replaced with sparse variants containing the sparsity information of their operands. This optimization leads to a huge increase in performance for simulating rigid body dynamics, while preserving the ability to write the derivation of the equations of motion in its most general form.

Stage 3: Structure recovery

There are two optimization criteria to take in mind when generating low level code. The first is obviously to try and generate fast code, but the second is to try and generate *small* code. Smaller code does not mean faster code, but it does mean a smaller memory footprint. This is especially important for micro-controllers where code size is a very real constraint. This stage of code generation is particularly concerned with generating small code and is essential for the implementation of central pattern generators on micro-controller type hardware (such as the *Salamandra robotica II* (Knüsel, 2013)).

After having destructured the model into simple pairs of {`DataTable` entry, expression}, we can try to recover some structure as a means of generating smaller code. The structure being recovered is that of common parametrized (sub)expressions. Two expressions are considered common when they can be made equal by means of parametrization of sub-expressions retrievable from the `DataTable`. To illustrate this, consider the following two assignments:

$$\boldsymbol{S}_i = \boldsymbol{S}_a + \boldsymbol{S}_b * \cos(\mathrm{pi} * \boldsymbol{S}_t) \tag{2.97}$$

$$\boldsymbol{S}_j = \boldsymbol{S}_b * \cos(\boldsymbol{S}_t * 5) + \boldsymbol{S}_c \tag{2.98}$$

where $\boldsymbol{S}$ is the `DataTable`, a, b, c and t are various elements in $\boldsymbol{S}$ and pi is a numerical constant. First, expressions are transformed into a canonical form. This is essential because it results in a unique representation of equivalent expressions, such that they can be compared in linear time for equivalence. Canonicalization essentially orders operands of commutative binary operators into a canonical order and canonicalizes equivalent operations (such as as unary minus and `-1*`). The canonicalized expressions are:

$$\boldsymbol{S}_i = \boldsymbol{S}_a + \boldsymbol{S}_b * \cos(\mathrm{pi} * \boldsymbol{S}_t) \tag{2.99}$$

$$\boldsymbol{S}_j = \boldsymbol{S}_c + \boldsymbol{S}_b * \cos(5 * \boldsymbol{S}_t) \tag{2.100}$$

To determine if the two expressions have a common, parametrized form, we can simply compare each individual instruction for parametrized equivalence. Two instructions are equivalent if

1. The instructions are strictly equivalent, exact equality, and
2. The instructions can be parametrized, i.e. passed as a function argument. For example, `DataTable` variables can be parametrized, but a binary operator cannot

During this procedure, value instructions (such as numerical constants) can be promoted to be stored in the `DataTable` if this means the expressions can be made common. In the example above this is what would happen, since pi and 5 are not equivalent, but can be promoted in which case they can be parametrized. The resulting parametrized function which can compute both expressions, and the transformed expressions are:

$$f(x_1, x_2) = x_1 + \boldsymbol{S}_b * \cos(x_2 * \boldsymbol{S}_t) \tag{2.101}$$

$$\boldsymbol{S}_i = f(\boldsymbol{S}_a, \boldsymbol{S}_{\mathrm{pi}}) \tag{2.102}$$

$$\boldsymbol{S}_j = f(\boldsymbol{S}_c, \boldsymbol{S}_5) \tag{2.103}$$

The reason for this common expression parametrization is two-fold. First, simply doing the transformation to capture common expressions in functions already reduces code size. At the same time, the compiler is still able to inline the function call if it decides to do so, i.e. there is no loss of performance. Secondly, it allows for automatic de-vectorization and loop generation of common expressions which drastically reduces code size.

Stage 4: Dependency resolution

To assist code generation, the next step is to generate a dependency graph of all the expressions that need to be computed. The objective is, given a set of states that need to be computed, 1) determine all required dependencies of those expressions (such as intermediate values) that need to be computed as well, and 2) determine the order in which all these expressions need to be computed.

Stage 5: Abstract program generation

Instead of directly generating C code, an abstract program is generated containing an abstract version of the code. This is useful because it allows for the abstraction of the specific code generator. Currently, the best supported code generator in *rawc* is the *C* code generator, but an experimental *JavaScript* exists as well.

Generating the program consists of separating computations into several stages of execution. Generally speaking, computations are separated into:

- *prepare*: this sets the pre-initialized state of the network and should be called just before *init*.
- *init*: initialize the network at time $t = 0$. This is called just after *prepare*. The separation of the two allows for variables to be set externally in between *prepare* and *init*.
- *diff*: computes the differential equations.
- *post*: computes any values that need to be observable after a successful integration step.

The abstract program also contains abstract versions of event handling, management of delayed expressions, updating of random values and user functions. Each time a set of expressions needs to be computed, its consecutive common (sub)expressions are automatically embedded inside an abstract loop such that minimal code can be generated.

Stage 6: Code generation

The final stage is the actual code generation. This stage is relatively easy since all the hard work has been done by generating the abstract program. What is left is to translate ċodγn instructions to *C*. Fortunately, the mathematical expressions in ċodγn are not that different from such expressions in *C*. Mathematical operations are translated to standard *C* implementations (where possible). Functions which are not supported in *C* (more specifically, in libm) are provided by a small *rawc* math library. This library is included directly in the generated code such that the *C* compiler can still inline these functions if it sees fit.

Micro-controllers do not always have all the required mathematical functions available, or standard implementations might be inefficient. For example, it is not uncommon to use a lookup table for functions such as sin and cos because their computations are expensive and take variable time depending on their arguments. *rawc* allows any math function to be overridden by user provided implementations.

The resulting code is generated in such a way that it can be compiled directly, without need for configuration or any external dependencies. It should be noted that the generated code currently only supports floating point arithmetic and is thus not suitable for micro-controllers without an FPU (floating point unit), or enough power to emulate floating point instructions.

2.7.4 Performance comparison

We briefly compare the performance increase due to the use of rawc when compared to libcodyn. We first look at our previous oscillator example. Recall that using libcodyn we were able to simulate up to 50 fully coupled oscillators in real-time. When we use rawc to generate optimized code for this network the simulations run approximately 130 times faster than real-time. The increase in performance does depend on the type of simulation, but it generating optimized code can result in up to two orders of magnitude improved performance.

We continue to compare the performance of the RBD simulation in codyn with two popular alternatives, Bullet and SimMechanics. We are mainly interested in practical time performance, i.e. how much real-time does it take to simulate a system of N degrees of freedom for T seconds. For each simulator, we measure how long it takes to simulate 5, 10, 20, 30, 40 and 50 degrees of freedom when simulating for 30 simulated seconds. The degrees of freedom are configured in a single, long chain of revolute joints, which is a worst-case system configuration for the method used in codyn to solve for the equations of motion.

We tried to make the comparison fair, by choosing the same simulation time step (1 millisecond) for all simulators, using fixed time step integrators. Importantly, we did not spend significant time trying to optimize for simulator settings. It may be possible that different performance can be obtained for each simulator by carefully tuning various (possibly problem dependent) parameters. However, here we focus on standard settings for each simulator. For SimMechanics, we used the Accelerated (i.e. compiled) model target with optimizations turned on.

All simulations were performed on an iMac, 3.2 GHz Intel Core i3 with 4GB of 1333 MHz DDR3 memory. Furthermore, we used Bullet 2.82, Matlab 2013a and codyn 3.6. Numbers shown here may vary depending on the platform and the version of the software used. Figure 2.17 shows the results of measuring performance of the four simulators. Here we can see that standard codyn does not perform particularly well, being barely realtime with 5 DOFs. Bullet and SimMechanics perform an order of magnitude better and show very similar performance. However, when we use codyn rawc to generate optimized code, we can see that (in particular for smaller systems), rawc outperforms both Bullet and SimMechanics by an order of magnitude. Finally, both Bullet and SimMechanics scale better with increasing number of DOFs when compared to codyn rawc, showing approximately similar performance for 40 DOFs and better performance for larger systems.

As noted before, the single chain of joints is a worst-case for solving the equations of motion using Recursive Newton Euler and the Composite Rigid Body algorithms. Whereas for both Bullet and SimMechanics the performance is mostly invariant of the system configuration, this is not so for codyn. In particular, performance of codyn depends largely on the number of branches in the kinematic tree. Another important step for improved performance when using rawc is the sparsity optimization stage (which is enabled by default). To illustrate to effect of branching and the sparsity optimization, we performed a second benchmark comparing the following four cases:

1. rawc with 1 branch and sparsity enabled

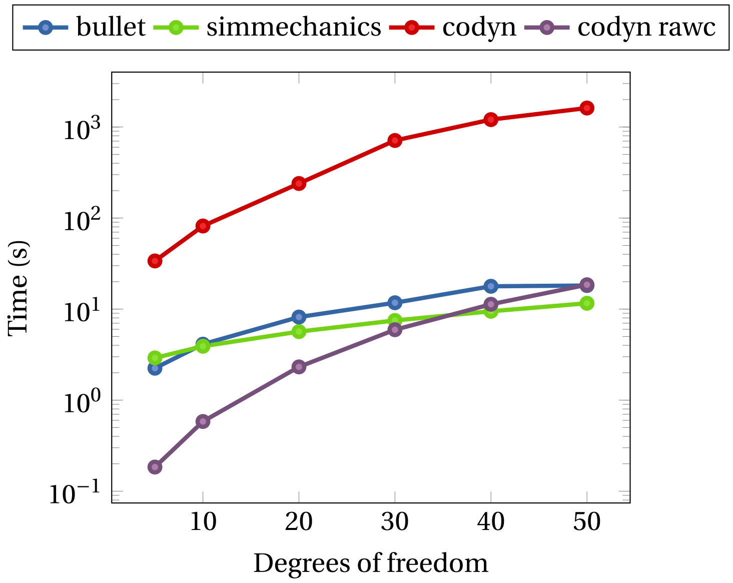


Figure 2.17 – Simulation performance of Bullet, SimMechanics, codyn and rawc on systems with increasing number of degrees of freedom. Simulations were performed for 30 simulated seconds. The y-axis shows in logarithmic scale the real time it took for the simulations to finish. As expected, codyn itself is an order of magnitude slower than both Bullet and SimMechanics (which show similar performance). When using rawc to generate optimized code, we however obtain another order of magnitude faster simulations, for systems up to 20 degrees of freedom than Bullet or SimMechanics. Both Bullet and SimMechanics show better scaling properties towards systems with a larger number of DOFs.

2. rawc with 1 branch and sparsity disabled
3. rawc with 5 branches and sparsity enabled
4. rawc with 5 branches and sparsity disabled

Note that case 1) corresponds to the case shown in figure 2.17. The degrees of freedom are distributed over the 5 branches, meaning that for example for a system with a total of 10 DOFs, there are 5 chains of each 2 revolute joints. Figure 2.18 shows the results of running the performance benchmark for each of the four cases. As can be seen, the difference between a branching factor of 1 and 5 is again an order of magnitude. Finally, it is shown that the sparsity optimization can significantly increase performance as expected.

2.8 Tools

The design of cȯdẏn as a core library with an accessible API (application programming interface) has enabled a rich tooling environment for analyzing, simulating and manipulating cȯdẏn models. Rather than an afterthought, cȯdẏn has been designed from the start as an easy to access core library enabling rich tooling. This is important because it allows third parties to

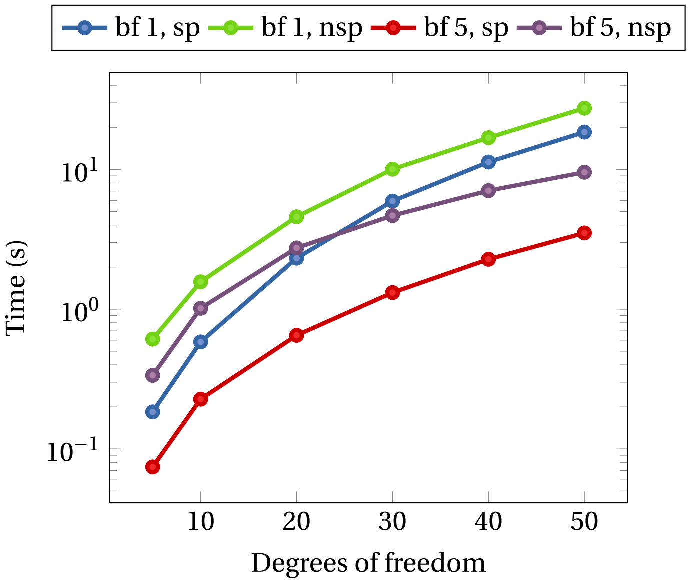


Figure 2.18 – Performance of codyn rawc comparing different branching factors (1 and 5) and the effect of sparsity optimization. bf 1 and bf 5 are respectively branching factors 1 and 5, and sp/nsp indicates respectively sparsity optimized and not sparsity optimized. The first case (in blue) is identical to the rawc case shown in figure 2.17. As shown, both the branching factor and the sparsity optimization have a large impact on general performance.

easily extend and modify functionality in cȯdyn, adding additional analysis or automation tools, and integrating cȯdyn in existing software frameworks.

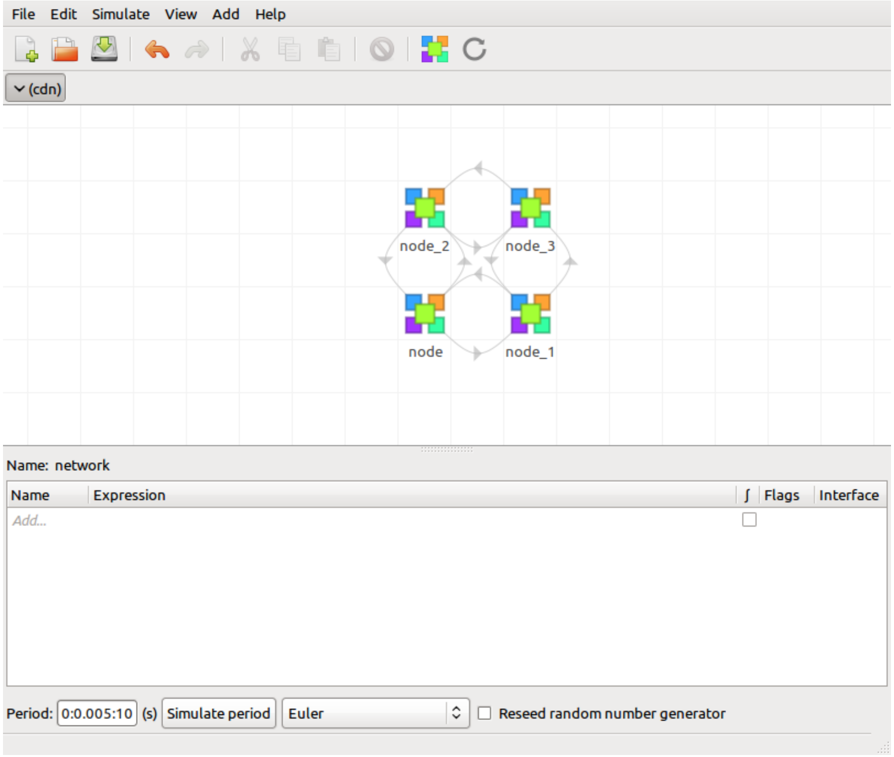
2.8.1 Command line tools

cȯdyn comes packed with a multitude of command line tools to analyze, simulate and render cȯdyn models. Additional tools are easy to develop externally since the underlying library has been designed with tooling in mind. The most important tools are presented below.

- `cdn-monitor`: a tool to quickly simulate and monitor certain variables in the network. The output is in the form of a simple tabular format which can be easily consumed by other tools for further processing
- `cdn-rawc`: the previously described tool which transforms a network into a high performance, real-time ready implementation without loss of functionality
- `cdn-compile`: a convenient tool to quickly compile and validate a network, providing extensive error reporting
- `cdn-render`: a tool that outputs a graphical rendering of a network in a variety of formats
- `cdn-repl`: a *Read-Eval-Print-Loop* interactive console for inspecting, evaluating and quick plotting of cȯdyn models

2.8.2 Graphical designer interface

Besides being a framework for high performance simulation of coupled dynamical systems, cȯdyn is also meant to be an educational platform. To introduce students to dynamical systems modeling and numerical integration, a graphical user interface is provided in which cȯdyn models can be constructed and inspected graphically. Furthermore, models can be directly simulated while monitoring variables. It provides a great way to experiment with simple models and explore the effect of model parameters, without the need to dive into the cȯdyn language. Figure 2.19 shows the basic graphical interface.



Name: network

Period: 0:0.005:10 (s)

Euler

☐ Reseed random number generator

Figure 2.19 – Screenshot of the cȯdyn graphical user interface. The canvas represents the cȯdyn network and can be interacted with to create and modify nodes and edges. Variables can be added, removed and inspected in the bottom panel.

2.8.3 Supported languages

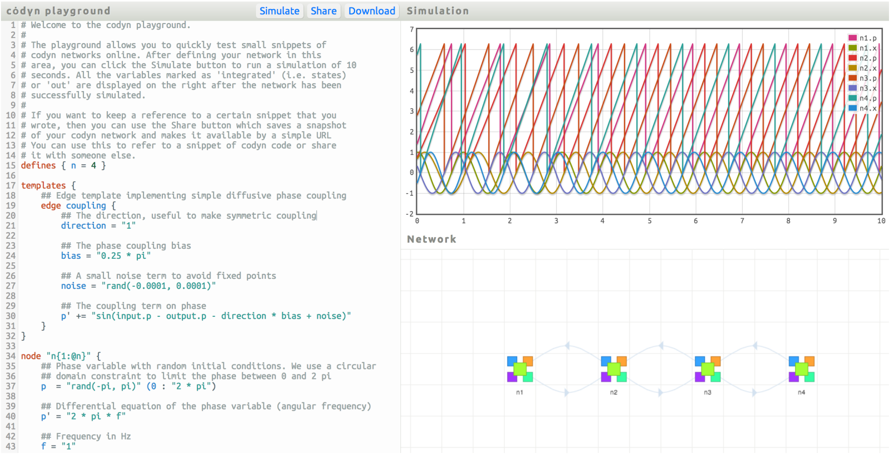
The core cȯdyn library is written in C. Although a higher level language would have made implementation of the feature rich cȯdyn platform easier, using C has been a conscious choice. Doing so has enabled cȯdyn tools to be written in a variety of different higher level languages since it is relatively straightforward to consume C based software libraries from other programming languages. cȯdyn is installed with excellent support for the popular Python and C# languages, which makes building tools on top of the core cȯdyn library very easy. Much of the more complicated tooling, such as the code generator (`cdn-rawc`) and the graphical designer interface have been written using the C# language support from cȯdyn.

2.9 Availability

Development of cȯdyn uses the git version control system running on the cȯdyn server. All sources can be directly viewed and obtained from the git server running at http://git.codyn.net/. Regular releases of all software in the form of tarballs is also available for download at http://download.codyn.net/releases/.

There are two important factors that determine whether or not a software framework is easy to adopt. The first is the availability of documentation, manuals and tutorials. Without proper documentation, it is difficult to use any reasonably complex software framework. All documentation for cȯdyn is available on the website. The second major factor is how low the barrier of entry is to get started. cȯdyn has full support for the GNU/Linux and OS X platforms. Packages for Ubuntu i386/x86_64 and OS X (>= 10.6) are available for download from the website (http://www.codyn.net/download.html) and provide all libraries and tools without the need for manual compilation or installation.

To lower the barrier even further, a simple online playground is available on the cȯdyn server allowing users to try out a limited version of cȯdyn directly from the browser at http://play.codyn.net/. All examples given in this chapter are available on this playground to try out and observe. Figure 2.20 shows a screenshot of the online playground.



```
# Welcome to the codyn playground.
#
# The playground allows you to quickly test small snippets of
# codyn networks online. After defining your network in this
# area, you can click the Simulate button to run a simulation of 10
# seconds. All the variables marked as 'integrated' (i.e. states)
# or 'out' are displayed on the right after the network has been
# successfully simulated.
#
# If you want to keep a reference to a certain snippet that you
# wrote, then you can use the Share button which saves a snapshot
# of your codyn network and makes it available by a simple URL
# You can use this to refer to a snippet of codyn code or share
# it with someone else.
defines { n = 4 }

templates {
    ## Edge template implementing simple diffusive phase coupling
    edge coupling {
        ## The direction, useful to make symmetric coupling
        direction = "1"

        ## The phase coupling bias
        bias = "0.25 * pi"

        ## A small noise term to avoid fixed points
        noise = "rand(-0.0001, 0.0001)"

        ## The coupling term on phase
        p' += "sin(input.p - output.p - direction * bias + noise)"
    }
}

node "n{1:@n}" {
    ## Phase variable with random initial conditions. We use a circular
    ## domain constraint to limit the phase between 0 and 2 pi
    p  = "rand(-pi, pi)" (0 : "2 * pi")

    ## Differential equation of the phase variable (angular frequency)
    p' = "2 * pi * f"

    ## Frequency in Hz
    f = "1"
```

Figure 2.20 – Screenshot of the cȯdyn online playground. The panel on the left shows the cȯdyn declarative language. A rendering of the structure of the current network is shown in the bottom right panel. Once simulated, resulting signals are automatically plotted in the top right panel for inspection. The cȯdyn network can be downloaded or easily shared online by obtaining a permalink to the playground document.

2.10 Conclusion

In this chapter we have presented a novel and open methodology for the design and modeling of coupled dynamical systems. The representation chosen by cȯdγn leads to a natural modeling structure of many types of dynamical systems, including coupled oscillators (e.g. central pattern generators) but also rigid body dynamics. A complete, state of the art implementation of Featherstone (2008) is provided, including various joint models, forward/inverse dynamics, contact modeling and closed loop dynamics. Importantly, there is no dedicated RBD engine in cȯdγn, and everything is structurally declared and derived using the cȯdγn language only. This allows for a unified representation of a system's dynamics, whether it be for control (such as central pattern generators) or rigid body dynamics, or other types of dynamics.

Furthermore, various tools built around the core cȯdγn library provide a multitude of useful functionality. Using `cdn-rawc`, efficient, optimized, low-level code can be generated automatically from any high-level representation of a cȯdγn dynamical system, without loss of generality or expressiveness. The resulting code is suitable for Real Time or embedded systems, such as those often used in robotics, or even micro-controllers. The `cdn-studio` provides a graphical user interface in which dynamical systems can be modeled in a graphical manner and experimented with, making it a useful tool for educational purposes. Together with its availability for GNU/Linux and OS X platforms, the website with documentation and instructions and the online playground, cȯdγn makes it easy to get started.

We will first use cȯdγn for the modeling of the rigid body dynamics of an adult sized human in chapter 4. Here we use the capabilities of `cdn-rawc` to generate fast code suitable for large scale, population based optimization. In the same chapter, we also implement a model of the CoMan humanoid robot in cȯdγn and see how the use of the hard contact model available in cȯdγn provides a more stable simulation of ground contacts, leading to a reduced objective complexity. Finally, in chapter 5 we use the closed loop dynamics modeling and powerful model parametrization of cȯdγn to simulate various wearable robot morphologies while optimizing for human locomotion assistance.

There are limitations to the way cȯdγn works as well. It is only suitable for the modeling of systems governed by ordinary differential equations. Furthermore, if a system cannot be easily represented by nodes and edges, then modeling it in cȯdγn can be, although doable, difficult. With a specific focus on Real Time and embedded systems, cȯdγn specifically targets dynamical systems which do not alter structure over the course of their simulation. When looking at the simulation of rigid body dynamics specifically, it is therefore ill-suited for the simulation of modular robots, to give an example, which reconfigure during operation. Although cȯdγn provides RBD, it does not at present provide any larger infrastructure for simulations, apart from numerical integration. This means, for example, that there is no interactive, graphical simulation environment, no abstraction of actuators, motors or sensors (such as cameras or range finders, etc.). In other words, it provides the bare simulation, but currently does not provide a fully integrated, robotics simulation environment.

Future work includes further increases in performance, specifically considering closed loop systems and contact models. Furthermore, the appropriate resolution of constraint systems such as the hard contact model can be improved as well and is currently unsuitable for handling large numbers of contacts and is limited currently to simple point contacts. It would also be interesting to extend the RBD features of cȯdγn towards a full robotics simulator package, possibly integrating it with existing frameworks such as Gazebo.

cȯdyn model 2.21 – Example of a closed loop pantographic leg [play]

```
include "physics/physics.cdn"
include "physics/cjoints.cdn"

integrator {
    method = "runge-kutta"
}

node "panto" : physics.system {
  node "hip" : physics.joints.revoluteY {
    com = "[0; 0; -0.02]"
      m = "0.01"
      I = "Inertia.Box(m, 0.01, 0.005, 0.04)"
      q = "0.1 * pi"

      # Torque due to slight damping
      D = "1e-3"
      τ = "-D * dq"
  }

  node "knee" : physics.joints.revoluteY {
     tr = "[0; 0; -0.04]"
    com = "[0; 0; -0.03]"
      m = "0.02"
      I = "Inertia.Box(m, 0.01, 0.005, 0.06)"
      q = "-0.4 * pi"

       # Torque due to a virtual spring around the
       #  initial angle of the leg
      q0 = "q" | once
       K = "0.1"
       τ = "K * (q0 - q)"
  }

  node "ankle" : physics.joints.revoluteY {
     tr = "[0; 0; -0.06]"
    com = "[0; 0; -0.025]"
      m = "0.01"
      I = "Inertia.Box(m, 0.01, 0.005, 0.05)"
      q = "0.4 * pi"
  }

  node "par" : physics.joints.revoluteY {
     tr = "[0; 0; 0.02]"
    com = "[0; 0; 0.025]"
      m = 0.1
      I = "Inertia.Box(m, 0.01, 0.005, 0.05)"
      q = "-0.4 * pi"
  }

  edge from "{hip,knee,ankle}" to ["knee", "ankle", "par"] : physics.joint {}

  # Close pantographic parallel structure
  node "parcl" : physics.cjoints.revoluteY {
    tr = "[0; 0; 0.06]"
  }

  edge from "{par,parcl}" to ["parcl", "hip"] : physics.cjoint {}

  include "physics/model.cdn"
  include "physics/dynamics.cdn"
}
```

3 Optimization

Apart from modeling dynamics, a second major cornerstone for the research presented in Part II is *optimization*. Just like ‘dynamics’, ‘optimization’ has a very broad definition. It is generally defined as the process of making *something* as *perfect* as possible; the execution of this process thus yields an *optimal* result. Optimization is a process that can be applied to any type of problem which has an associated *cost*. This cost, as a function of a solution to a problem, is what determines what the perfect solution to a problem is, namely that solution which renders a minimal cost. When the solutions yielding a minimal cost are not known beforehand, then optimization processes can provide a powerful framework to find or discover them.

Optimization is such a large topic of research that it squarely falls outside of the scope of this text to discuss it fully. Nevertheless, optimization is an ubiquitous method used extensively in robotics and during the work presented in this thesis as well, and as such it deserves a small introduction. There are different ways to create a taxonomy of optimization methods. In the field of robotics, which is concerned with mechanical design and control of articulated structures, the methods being applied can usually be divided on the *metaheuristic* axis.

The general problem of optimization can be formulated as

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f_i(\boldsymbol{x}), (i = 1, \cdots, i = I) \tag{3.1}$$

$$\text{such that} \quad h_j(\boldsymbol{x}) = 0, (j = 1, \cdots, j = J) \tag{3.2}$$

$$g_k(\boldsymbol{x}) \leq 0, (k = 1, \cdots, k = K) \tag{3.3}$$

i.e. find a solution for $\boldsymbol{x}$ which minimizes f_i (the cost function) with the equality constraints h_j on $\boldsymbol{x}$ and inequality constraints g_k on $\boldsymbol{x}$.

The use of metaheuristics refers to the search for and discovery of solutions by a procedure of informed and repeated trial-and-error, and does not require any knowledge of the behavior of f_i, h_j or g_k. This contrasts with classical methods, such as iteration methods (e.g. Newton’s method), which need information on f_i (usually its derivative) to determine in which directions of $\boldsymbol{x}$ an improvement of f_i can be obtained (within constraints imposed by h_j and

g_k). Optimal control (Zhou *et al.*, 1996) and multiple shooting are popular non-metaheuristic methods in the robotics community. The main advantage of using metaheuristics instead of these methods is that little knowledge of the cost function landscape is required. It therefore also allows for a more explorative search of the solution space since cost functions can be defined in very generic terms, whereas non-metaheuristic methods usually require more specific costs. Consequently, metaheuristic methods usually have less problems with local optima.

Metaheuristic methods also have disadvantages. First, heuristic methods are often *stochastic* in nature. This means that to get reliable results, optimizations often have to be repeated a number of times. The optimization process itself can also be very computationally expensive, since search, although informed, is still a process of trial-and-error, often requiring many trials. Metaheuristics also often lack fundamental underlying theory and do not guarantee that an optimum is found in finite time.

A large number of metaheuristic methods are inspired by natural processes. Whether they are evolutionary processes (Genetic Algorithms, (Goldberg *et al.*, 1989)), swarming behavior (Particle Swarm Optimization, (Kennedy and Eberhart, 1995)) or methods derived from studying behavior in colonies (Ant or Bee Colony Optimization, (Dorigo and Birattari, 2010; Karaboga and Basturk, 2007)), they are all inspired by observations of natural processes. The idea that we can optimize engineering problems by mimicking natural processes is a powerful one. Indeed, it is widely accepted that these processes lead to (local) adaptation to obtain an optimal *fitness* (the inverse of cost). Most of the methods derived from these observations are based on a populace with such internal dynamics that over time, a global optimum can be found.

This chapter contributes a novel particle swarm optimization (PSO) based method, suitable for the simultaneous optimization of solution structure and its parameters. This algorithm is used in chapter 5 for the co-design of a lower limb, assistance providing wearable robot. Furthermore, a general architecture and software framework for large scale optimizations using population-based methods has been developed and is provided and distributed under an open source license.

We now first begin with a brief introduction into population-based optimization methods. Thereafter the novel PSO algorithm will be explained in detail. Section 3.3 continues to discuss the application of the optimization of multiple objectives, with a particular focus on population based methods used in chapters 4 and 5. Finally, section 3.4 describes a framework for performing large scale optimization that has been used for the work presented in Part II.

3.1 Population-based methods

Population-based optimization methods are based on maintaining a (possibly large) population of potential solutions to a particular optimization problem. These solutions are evaluated to obtain their objective fitness values. Based on this objective fitness, a new population is generated through a variety of mutation and recombination methods which generates new solutions to be evaluated. This process is then repeated until some stopping criterion is met.

Instead of relying on a rigorous treatment of the problem dynamics, these methods often rely on *heuristics* and *stochastic* processes to explore and discover solutions to a particular problem. They therefore usually do not guarantee to obtain a globally optimal solution. On the other hand, they can be used on problems for which the task dynamics are not well known, for which fitness landscapes are rough and unpredictable, and generally to explore large parameter and solution spaces without a-priori knowledge.

Population-based methods are therefore often successfully used to perform exploratory searches on open-ended problems. Due to the fact that often large populations need to be used to obtain satisfactory results, they are particularly well suited for large scale off-line optimizations, rather than on-line optimizations.

In Part II, we use population-based methods for the optimization of tasks related to human locomotion for especially these reasons. There exist a large number of these type of optimization algorithms and we briefly discuss the most popular ones here.

3.1.1 Genetic Algorithms

The genetic algorithm (Goldberg *et al.*, 1989) is perhaps the most classically regarded population-based optimization method. Inspired by naturally occurring evolutionary processes, in genetic algorithms a selection algorithm determines which *individuals* in the current population are considered fit for *breeding* offspring constituting the next generation. Mutation and cross-over operations provide for exploration of the parameter space and diversity of the population.

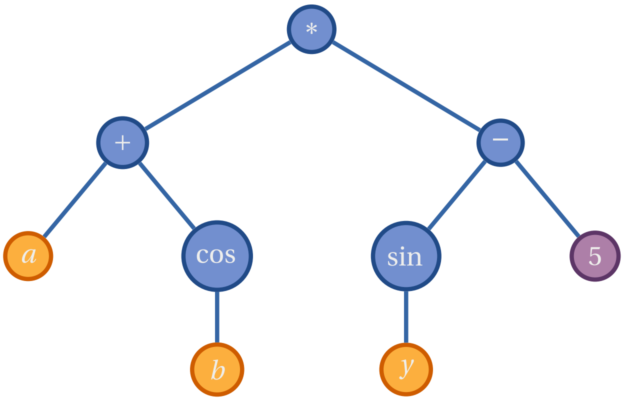
Variations of genetic algorithms differ in their choice of selection mechanisms of which there are many. Popular choices include tournament selection, roulette wheel selection and elitism. Selection methods can also be combined to create new selection methods with certain advantages on particular problem domains. Furthermore, different ways of generating the offspring lead to a large variety of genetic algorithms.

One of the difficulties of using genetic algorithms is the fact that there are so many different variants and choosing one that works well for a specific problem is not obvious. Additionally, each variant has a non-trivial amount of parameters which need to be set and which can significantly influence the performance of the algorithm.

3.1.2 Genetic Programming

Genetic programming (Koza, 1992) is a method for constructing task solving programs. The algorithm finds its roots in genetic algorithms, but instead of optimizing the parameters of a parametrized problem, it optimizes the structure of a program whose purpose it is to solve the task. It can therefore be applied to problems for which the structure of the solution is not known and which are thus hard to parametrize.

In genetic programming, programs are usually represented in tree-like structures and, similar to genetic algorithms, a set of mutation and cross-over operations manipulate these trees to obtain new sets of programs.



$$\left(a+\cos(b)\right)*\left(\sin(y)-5\right)$$

Figure 3.1 – Representation of a solution program generated by genetic programming for the canonical problem of mathematical function fitting. Mathematical operators, variables and numerical constants make up the alphabet of the genetic program. The resulting mathematical expression represented by the program can be evaluated to obtain the quality of the solution.

As an example, consider the textbook case of fitting a multi-variate mathematical function to a set of data (i.e. data modeling). The building blocks of the programs generated by genetic programming are mathematical operations, variables and numerical constants. The goal is then to construct programs (mathematical functions) which best explain the data. Figure 3.1 shows a canonical representation of a solution program as generated by genetic programming for this problem. The generated program can be evaluated based on how well it explains the data. From this, new programs are constructed using structural mutations (additions of new nodes, removal of existing nodes) and by combining solutions through cross-over operations to obtain a new set of candidate solutions. This process is repeated until a sufficient explanation of the data is found.

The above example is one of the canonical applications of genetic programming, but certainly not the only one. It has been successfully used for a variety of computer science problems such as subroutine discovery (Rosca and Ballard, 1996) or uses in quantum computing (Spector *et al.*, 1999). It has also found applications outside of the computer science domain, such as for the design of components with specific constraints (Lohn *et al.*, 2005).

Although genetic programming is a versatile and useful technique, it also has several disadvantages. For example, it can be hard to constrain the solution space of genetic programs. Multiple solutions with different internal structures can still perform the exact same computation. Furthermore, just as with genetic algorithms, there are a possibly large number of algorithm parameters to set.

3.1.3 Particle swarm optimization

Particle swarm optimization is in many ways an answer to the difficulties of using genetic algorithms in practice. Particle Swarm Optimization (from here on referred to as PSO) is another population based, stochastic optimization algorithm which has been a popular alternative to genetic algorithms since it was first introduced in (Kennedy and Eberhart, 1995). In its essence, PSO is a very simple algorithm, consisting only of two simple equations which govern its dynamics. Conceptually, the PSO is a cooperative algorithm where the individual particles share information about known solutions of the particular problem being solved. Shi and Eberhart (1998) have proposed a slightly modified version of the original PSO algorithm, which is often the algorithm used today when referring to PSO. The two equations describing the whole algorithm are given in equation 3.4.

$$\begin{aligned} v_i(t+1) &= w \cdot v_i(t) + r_{i1} \cdot c_1 \cdot (X_i - x_i(t)) + r_{i2} \cdot c_2 \cdot (X_g - x_i(t)) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \end{aligned} \tag{3.4}$$

Here x_i is the current *position* of particle i in the parameter space. It is thus the vector of real-valued parameter values representing a particular solution to the problem being solved. v_i is the current *velocity* of particle i. Furthermore, r_{i1} and r_{i2} are two random numbers uniformly distributed between 0 and 1, X_i is the best solution as found by particle i (its personal best) and X_g is the global best known solution. The constants c_1 and c_2 determine the importance of respectively local versus global search. Compared to the original algorithm as described in Kennedy and Eberhart (1995), an additional term is introduced, the so called inertia factor w (Shi and Eberhart, 1998). The purpose of w is to improve the convergence by smoothing the parameter space and has been generally found to improve the performance of the PSO.

The algorithm as presented has only three parameters, the inertia factor w and the two constants c_1 and c_2. Although research has been done as to the importance and influence of these parameters, they are usually (unless explicitly researched) set to the values 1.494 for both c_1 and c_2, and 0.729 for w which can be shown to guarantee convergence of the algorithm (Clerc, 1999; Eberhart and Shi, 2000; Clerc and Kennedy, 2002).

Finally, to perform the optimization, an initial population of particles is generated each with an initial position vector x_i and initial velocity vector v_i. Both of these vectors are usually initialized such that they are randomly, uniformly distributed in a bounded parameter space.

We limit the maximum value of each dimension of the velocity vector to a fraction of the distance from the minimum parameter boundary to the maximum parameter boundary (Eberhart and Shi, 2000). This has been shown to give good results in general as there is more exploration (in particular in the beginning of the optimization). After the population has been initialized, at each iteration the fitness of each particle for the parameters x_i is calculated and X_i and X_g are updated accordingly. Then, for each particle, the particle's velocity v_i and x_i are updated using equation 3.4. The stopping criterion is often chosen to be a fixed number of iterations or some measure of convergence.

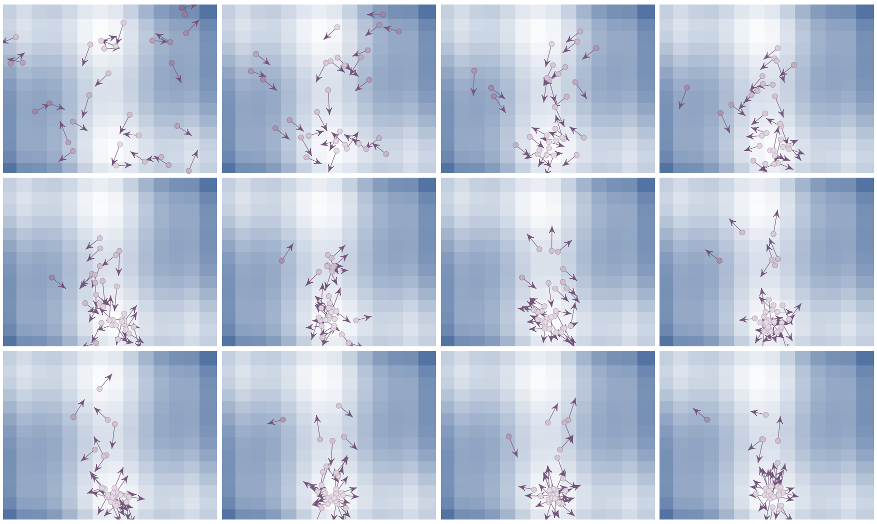


Figure 3.2 – PSO optimization of the Six-hump camel function as defined in equation 3.5. This function has two global optima, at approximately the top and bottom center of the space shown here. The particles start out with random initial positions and velocities in the 2D parameter space. Particles then start to explore the parameter space based on their local and global best known parameters. As the iterations progress (left to right, top to bottom), particles start to converge on one of the global minima of the function.

Figure 3.2 shows an example of PSO optimizing a well known optimization test function, called the Six-hump camel function, defined as

$$f(x,y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + (-4+4y^2)y \tag{3.5}$$

We look at the parameter space bounded by $x \in [-2,2]$ and $y \in [-1,1]$, which contains two global minima in this range, at $(0.09,-0.71)$ and $(-0.09,0.71)$. As shown in figure 3.2, PSO is able to converge on one of the global minima. This also shows how PSO will only finally converge on a single optimum, and which one depends on the initial conditions.

3.2 Metamorphic particle swarm optimization

Since its original inception, the Particle Swarm Optimization algorithm (or PSO) (Kennedy and Eberhart, 1995) has seen a considerable amount of attention in the evolutionary computation community. Partly due to its simplicity and elegance, since then many new varieties of PSO have been developed by researchers in the community trying either to address some of its shortcomings (such as stagnation (Worasucheep, 2008; Evers and Ghalia, 2009), diversity (Monson and Seppi, 2006) or niching (van den Bergh and Engelbrecht, 2004; Nickabadi *et al.*, 2008)) or to improve its performance tailored towards specific sets of problems (such as multiple objectives or constraints (Ray and Liew, 2002; Parsopoulos and Vrahatis, 2002; Leong and Yen, 2008)).

We now turn to look at solving such a particular set of problems: Namely, the optimization of a problem for which a discrete set of solution classes exists , each with a (possibly overlapping) subset of continuous parameters taken from the total parameter set. The optimization then needs to take into account the discrete problem as well as optimizing the continuous parameters used for this particular solution class. This might seem like an abstract problem, but indeed, many real problems are formulated this way. Often we choose to optimize each solution class of the problem independently or even manually and compare the results later. This however is 1) not practical for problems for which a large set of different solutions exist, 2) inefficient for problems with a large possible solution set but a small probable solution set, and 3) it can be biased by human intervention especially for the cases for which human intervention is not sufficient. To solve these kinds of problems we require an algorithm which 1) makes informed, discrete decisions about which classes of solutions to explore and 2) finds optimal parameter values for these classes of solutions.

The first contribution to solving discrete binary problems using PSO came not long after the original PSO algorithm was published. In Kennedy and Eberhart (1997), the original author of PSO details a version of PSO which uses probabilities of a discrete value switching from 0 to 1 (or the other way around) instead of the actual values as the parameters being optimized. More recently, this approach was generalized in Clerc (2004) in which the definitions and operations of the PSO (position, velocity, subtraction, external multiplication and move) are redefined for the discrete domain. An extension of the original binary discrete PSO algorithm was presented in Pugh and Martinoli (2006) in which discrete multi-valued problems are solved by adding a probability for each possible value that the discrete variable can take. Here we take inspiration from this work and use a similar approach to making discrete choices by using probabilities. However, unlike in previous approaches we will define probabilities related to exploration and exploitation similar to those used in PSO to search the set of discrete solution classes while at the same time solving the continuous problem in each solution class. We see that this allows for more control of the way the problem is solved while at the same time reusing concepts from the continuous domain, which have worked well in general, to the discrete domain. Although genetic algorithms and genetic programming could be used in a similar way to provide the discrete part of the optimization, it has been shown in Bourquin *et al.* (2004)

that PSO performs better for the type of locomotion optimizations that we are interested in. We are therefore interested here in reusing the collaborative/cooperative nature of the PSO in the discrete part. The novel algorithm that we designed for this particular problem is called Metamorphic Particle Swarm Optimization.

The following sections first describe the main MMPSO algorithm in detail. After this description we show some of the MMPSO properties on an example problem. Finally, we discuss some of the applications of the algorithm and future work.

3.2.1 Metamorphic PSO Algorithm

We found PSO to be a well performing and easy to understand algorithm for a wide variety of optimization problems. It often outperforms algorithms such as genetic algorithms (Ou and Lin, 2006; Latiff *et al.*, 2007) in a variety of different domains it has been applied to. The elegance of the algorithm, the small number of parameters (c_1, c_2 and w) to tune and the general performance are arguably some of its most prominent features.

The base PSO algorithm as described in the previous section works on continuous parameters. What we are interested in, in this work, however is a combination of a discrete set of parameter subspaces and a simultaneous optimization of each of these parameter subspaces (in which the parameters are continuous). We have coined our algorithm Metamorphic due to the fact that the it meta-optimizes the possible solution subspaces by **morphing** particles from one subspace to another, reconfiguring its parameter space. Note that here we do not mean *meta optimization* which is concerned about optimization of algorithm parameters or objective functions.

We briefly describe a concrete robotics problem (explained in more detail in section 3.2.3) to illustrate for which type of problems MMPSO was designed. Let us assume a certain robotic structure with K degrees of freedom, for which we want to find control laws for locomotion. Furthermore, let us assume that we can control each of these DOFs with three different modes of control, namely 1) oscillation, 2) continuous rotation or 3) a locked constant offset. We now have three choices of control modes to make for each of the K degrees of freedom. Instead of making these choices manually, we designed MMPSO to explore combinations of control modes for each DOF automatically. We will occasionally refer to this application of MMPSO in explaining certain concepts of the algorithm.

Concepts and Terminology

The Metamorphic PSO Algorithm (hereafter referred to by MMPSO) has been specifically designed for the type of problem described above. Still staying in the abstract domain, consider the following problem containing 9 parameters to optimize as shown in figure 3.3.

This schematic representation of the parameter space consists of three entities (**A**, **B** and **C**)

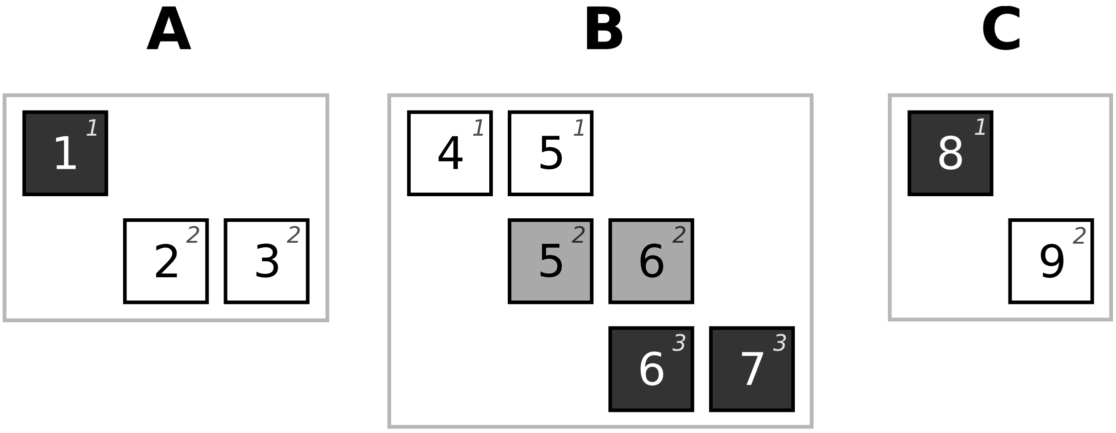


Figure 3.3 – Example parameter configuration of a single particle. Each of the parameter *pools* **A**, **B** and **C** depict a discrete number of parameter *groups*. The group number is indicated in the superscript of each box as well as by the background shading for clarity. In each pool, only one group can be active and optimized at a given time. Parameters can overlap between different groups as can be seen in pool **B**, where a valid set of parameters is either (4, 5), (5, 6) or (6, 7). One complete *subspace* is composed of selecting one group for each pool, for example {(1), (4, 5), (9)}. There are a total number of 9 parameters in this example.

which we call parameter pools. A parameter pool in MMPSO is something which defines a distinct number of possible parameters groupings active at a single given time. Thus, referring to figure 3.3, in pool **A** only either parameter (1) or parameters (2, 3) are active. In the context of MMPSO, we call these different parameter groups and in the text we indicate a group within parentheses (). The groups within each pool are mutually exclusive. Although the groups are mutually exclusive, the parameters in each group need not be. Indeed, as shown in pool **B** in figure 3.3, parameter 5 is active both in group 1 and in group 2. Similarly, parameter 6 is active in both group 2 and in group 3 (groups are indicated by a superscript in each box).

We have until now only explained the concepts of pools and groups. We still need to outline the concept of parameter subspaces. Given the definitions of the pool and group above, a parameter subspace is one, valid combination of groups chosen from each pool. In the text we will indicate subspaces with braces {}. In figure 3.3 possible subspaces are $\{(1),(6,7),(8)\}$ or $\{(1),(4,5),(9)\}$. The total number of possible subspaces results from simple combinatorics on the groups in each pool. In our example, the total number of subspaces would thus be $2 \times 3 \times 2 = 12$.

To relate the MMPSO parlance to our concrete robotics example, each DOF is represented by a pool and each control mode is represented by a group. Thus each pool contains three groups (oscillation, rotation, locked) to choose from. A particular subspace is then a specific combination of control modes for each DOF. Note that unlike depicted in figure 3.3, each pool here has the same configuration. The continuous parameters themselves are the control parameters corresponding to each control mode (such as oscillation amplitude and offset).

The goal of MMPSO is to efficiently search for solutions within these subspaces, dividing effort spent in each subspace based on a similar principle of collaboration as used by the base PSO algorithm. To accomplish this we separate the algorithm in two layers.

The inner layer

The *inner* layer is defined as one instance of a subspace (i.e. there are 12 distinct inner layers in our abstract example). Each inner layer runs an **independent** base PSO algorithm. Particles initially are equally distributed over the different subspaces (note that there can be more subspaces than particles in which case some subspaces remain initially unpopulated). Although we use the base PSO algorithm as defined in section 3.1.3, it should be noted that any extension or variant of PSO could be run without modification in the inner layer. The main contribution of MMPSO is the way particles are transferred between subspaces in what we call the *outer* layer.

The outer layer

The outer layer is a separate algorithm outside the inner layers responsible for migrating particles from one subspace to another. Figure 3.4 shows a schematic representation of the two-layered system. Each subspace contains a separate PSO and the outer layer migrates particles between subspaces. A subspace best solution, X_s is maintained in each subspace and is the equivalent of the globally best known solution X_g in the base PSO algorithm. We also introduce a new X_g which represents a new globally best known solution known only to the outer layer algorithm.

To transfer particles between subspaces we borrow the concept of the mutation operation from genetic algorithms. The basic idea is to *migrate* a particle from one subspace to another subspace based on migration probabilities. However, unlike in GA where a beneficial mutation is automatically propagated to the next generation, we do not have such a concept in our PSO. Simply moving particles from one subspace to another randomly chosen subspace will not provide appropriate pressure to explore subspaces that have a higher overall fitness more than other subspaces, since the particles are moved (and moved again) randomly.

To address this issue, we take inspiration from the concepts of local versus global search and exploration versus exploitation from PSO and introduce three migration probabilities. The *exploration migration probability* P_e, defines the probability of a random migration of the active group within each pool. The *local migration probability*, P_l, defines the probability of migrating to the group within each pool which is contained in the best solution known to that particle. Similarly, the *global migration probability*, P_g, defines the same type of migration probability as the local migration probability, but towards the globally best known solution X_g over all subspaces.

Together, these three migration probabilities will govern the search of the different subspaces

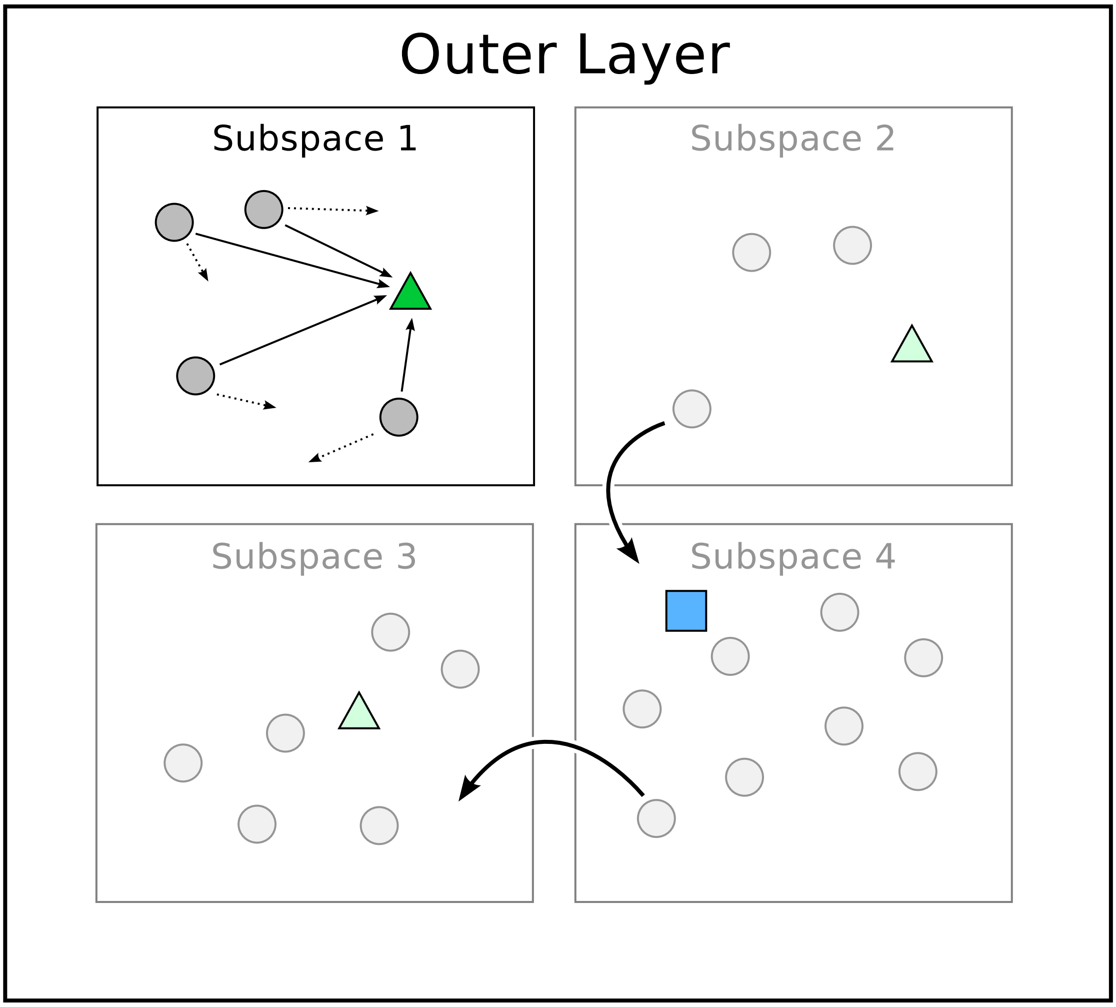


Figure 3.4 – Schematic overview of the two-layered algorithm. Each of the subspaces contains an independent PSO with a population set to the particles which are currently in the subspace. The green (triangle) particle represents the best known solution for each subspace which we call X_s (this is equivalent to X_g in the base PSO). The blue (rectangle) particle represents the globally best known solution taken over all the subspaces and is only known only to the outer layer algorithm. We call this solution X_g.

in a collaborative manner similar to how PSO tries to optimize parameters within a subspace. We can now define the probability $P(s_c \to s_j | s_c \neq s_j)$ of each particle, migrating from the current group (c) of a pool (s) to a group (j) different from c as given in equation 3.6.

$$\begin{aligned}
P(s_c \to s_j | s_c \neq s_j) &= 1 - (1 - \frac{P_e}{N-1}) \cdot (1 - P_l | s_j = s_l) \cdot (1 - P_g | s_j = s_g) \\
P(s_c \to s_j | s_c = s_j) &= 1 - \sum_{k}^{N} P(s_c \to s_k | s_c \neq s_k) \\
P_e + P_l + P_g &\leq 1
\end{aligned} \tag{3.6}$$

Here the notation $P(a \rightarrow b | a \neq b)$ is used to mean the probability of a transitioning to b given that b is different from a, thus the probability of a particle migrating from a particular group to a different group. This probability is calculated from the probabilities P_e, P_l and P_g as defined above and N is the number of different parameter groups in the pool s. Furthermore, s_l is the parameter group l of pool s in which the locally best known solution of the particle has been found and s_g is the parameter group g of pool s in which the globally best known solution (over all parameter subspaces) has been found.

Equation 3.6 proceeds to calculate first the probability of not migrating, which is given by the product of the probabilities of not migrating due to, respectively, exploration (P_e), local migration (P_l) and global migration (P_g). The probability of not exploring is given by 1 minus the probability to migrate according to P_e to any other group, of which there are $N-1$. Secondly, the probability of not migrating towards the locally known best group can be calculated by 1 minus P_l, given that the group to be transitioned to (s_j) is the locally best known group (s_l). We have adopted the notation $P_l | s_j = s_l$ here to evaluate to P_l when $s_j = s_l$, or 0 otherwise. The probability of not migrating towards the globally best known group is calculated in the same way. Finally, the resulting probability $P(s_c \rightarrow s_j | s_c \neq s_j)$ is then given by 1 minus the total probability of not migrating.

For completeness, the second equation provides the probability of staying in the same group (i.e. not migrating at all). This probability is simply 1 minus the total probability of migrating to any of the N other groups. In practice, only the first equation is used to calculate whether a group needs to be migrated. Finally, to guarantee proper probabilities, the sum of P_e, P_l and P_g must be smaller or equal to one.

The probabilities as described in equation 3.6 are proper probabilities in the sense that the sum of all the probabilities equals to 1 (this can be easily seen since the probability of **not** migrating is defined as 1 minus the sum of probabilities of migrating to a different group). They are also defined correctly such that setting for example $P_e = 0.5$ will cause on average one particle per two iterations to migrate to each pool randomly.

As an example, figure 3.5 shows schematically the migration probabilities involved for a given state of the pool $\mathbf{B} \in s$ for a particular particle (as shown before in figure 3.3). The figure portrays the case where the current group of $\mathbf{B}$ (B_c) is group 1, or parameters (4, 5). The locally best known group (B_l) is group 2, or parameters (5, 6) and the globally best known group (B_g) is group 3. There are then three probabilities $P(B_1 | B_1 = B_c)$, $P(B_2 | B_2 \neq B_c)$ and $P(B_3 | B_3 \neq B_c)$ which respectively represent the migration probability of 1) not changing the current group, 2) changing the current group from B_1 to B_2 and finally changing the current group from B_1 to B_3. Using equation 3.6, these probabilities then become as shown in equation 3.7. We will discuss ways to choose P_e, P_l and P_g to design certain behaviors of the algorithm in section

3.2.2.

$$
\begin{aligned}
&P(B_1 \rightarrow B_2) = 1 - (1 - \frac{P_e}{2}) \cdot (1 - P_l) \\
&P(B_1 \rightarrow B_3) = 1 - (1 - \frac{P_e}{2}) \cdot (1 - P_g) \\
&P(B_1 \rightarrow B_1) = 1 - P(B_1 \rightarrow B_2) - P(B_1 \rightarrow B_3)
\end{aligned} \tag{3.7}
$$

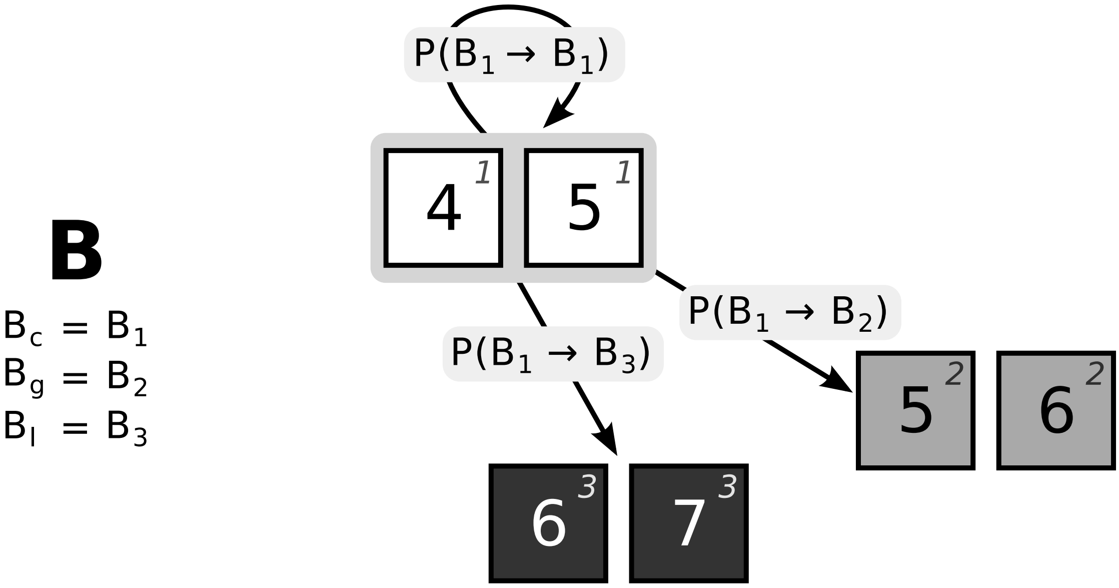


Figure 3.5 – An example of the migration probabilities involved in migrating the pool $\mathbf{B} \in s$ from one particular current group (B_c) to each possible group of **B**. The probabilities $P(B_1 \rightarrow B_1)$, $P(B_1 \rightarrow B_2)$ and $P(B_1 \rightarrow B_3)$ can be calculated using equation 3.6. The resulting probabilities (as functions of P_e, P_l and P_g) are given in equation 3.7.

Pseudo Code

A very short and concise pseudo code listing for the algorithm is given in algorithm listing 1 [1]. In short, at each iteration, a base PSO is run for each currently non-empty subspace. After this, the best local and global group (s_l and s_g) for each pool are updated according to the fitness of each particle. Finally, particles are migrated from one subspace to another by changing the group in each pool according to the probabilities P_e, P_l and P_g.

3.2.2 Properties

There is only one set of parameters left for the user of MMPSO to choose. These parameters are the mutation probabilities P_e, P_l and P_g. The values of these parameters are important

---

[1] A fully working example of the MMPSO algorithm implemented in matlab is available at: http://biorob2.epfl.ch/~jvanden/mmpso/mmpso_code_nicso_2013.zip

Algorithm 1 MMPSO

```
Subspaces: the set of all subspaces
Pools: the set of all pools
$P$: probability function of $s_c \rightarrow s_i$ with $P_e, P_l, P_g$

function MMPSO
    Particles ← initializePopulation

    while stopping condition not met do
        for $u \in$ Subspaces, $u \neq \emptyset$ do
            PSO(Particles $\cup$ $u$)                          ▷ base PSO on particles in $u$

        for $s \in$ Pools do
            $\{s_l, s_g\}$ ← updatePoolBest($s$)                 ▷ update $s_l$ and $s_g$

        for $p \in$ Particles do
            for $s \in$ Pools do
                migratePool($p$, $s$, $P(s_c \rightarrow s_i | s_c, s_l, s_g)$)   ▷ migrate $s_c \rightarrow s_i$ using $P$
```

since they will completely govern the behavior of the outer layer algorithm. As such, they need to be chosen carefully.

In general we would normally like to stimulate exploration early in the optimization, so the various subspaces are explored sufficiently and general (sub)optima can be located. As the optimization progresses, particles should start to focus more on their locally best known subspaces to explore these in more detail. Finally, particles should start to converge on the globally best known subspace to maximally optimize for that particular space during the late phases of the optimization process.

To get this kind of behavior, we can design the mutation probabilities using probability curves as functions of the number of iterations. Note that we assume here a stopping criterion based on the maximum number of iterations. If a measurable convergence criterion is used, then the probability curves can be a function of the convergence instead, however we have not explored this possibility yet. Figure 3.6 shows one particular choice of the probability curves. Here we used sigmoid shaped functions for the exploration and global exploitation probabilities, and a Gaussian shaped curve for the local exploitation probability. Choosing the shapes of the curves similarly to the ones shown in figure 3.6 generally works well.

Although we do not have a rigorous design methodology for the choice of these probabilities, we have developed an emperical and procedure to choose initial values for the probabilities. The basic procedure is to first estimate for how many iterations, on average, you want particles to explore a give subspace. This is very much problem dependent and the usual procedure is to perform a few optimizations and look at the fitness progression to see how many iterations it takes for particles to get a sense of how well a subspace can solve the problem. Then, given the number of particles, probabilities can be choosen such that particles explore on average that number of iterations in each subspace. It should be noted that this procedure is very much a manual process, and a certain knowledge about the problem domain has to be assumed (i.e. how difficult is the optimization process). Emperically, we found that obtained results are not very sensitive to the exact choice of the probabilities, but we have not yet done extensive studies to quantity this findings. Furthermore, future work includes automatic tuning of these probabilities based on estimations of convergence, which should lead to better automatic exploration and exploitation behavior without needing to manually choose the correct probabilities beforehand.

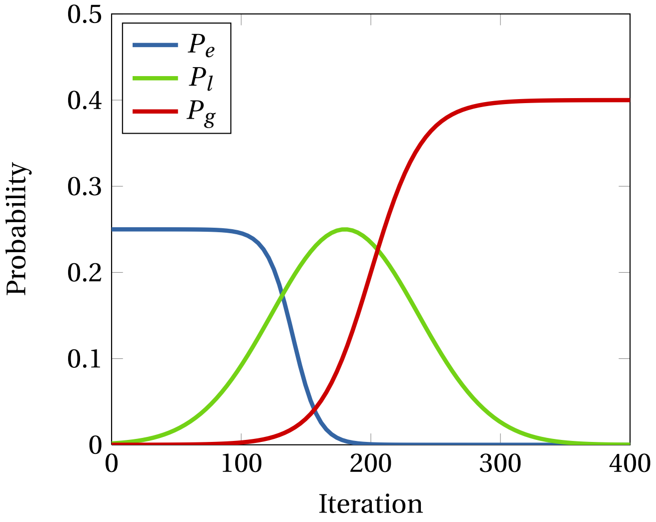


Figure 3.6 – Mutation probability characteristics for the exploration probability P_e, local exploitation probability P_l and global exploitation probability P_g, emphasizing early exploration and late convergence.

Example

In this section we will briefly show some characteristics of the MMPSO on the most simple numerical problem. Although the example is a trivial one, it makes it equally trivial to analyze its behavior. In this simple example we are going to consider only two parameters, x and y, both bounded in $[0, 1]$. We define one pool containing two groups. The first group is (x) and the second is (x, y). Thus a particle either optimizes for only x or both x and y. We further define two objective functions. The first is evaluated for particles optimizing $\{(x)\}$ and the objective is simply x itself, with a maximum value of 1. The second objective function is evaluated for $\{(x, y)\}$ and is given by $2-(|x-0.5|+|y-0.5|)$, which has a maximum value of 2 at $x = 0.5$ and $y = 0.5$. These objectives were chosen such that the maxima in both subspaces are at different values of x, to illustrate the ability of the algorithm to find both.

The population size in this example is set to 40 particles and the optimization lasted for 70

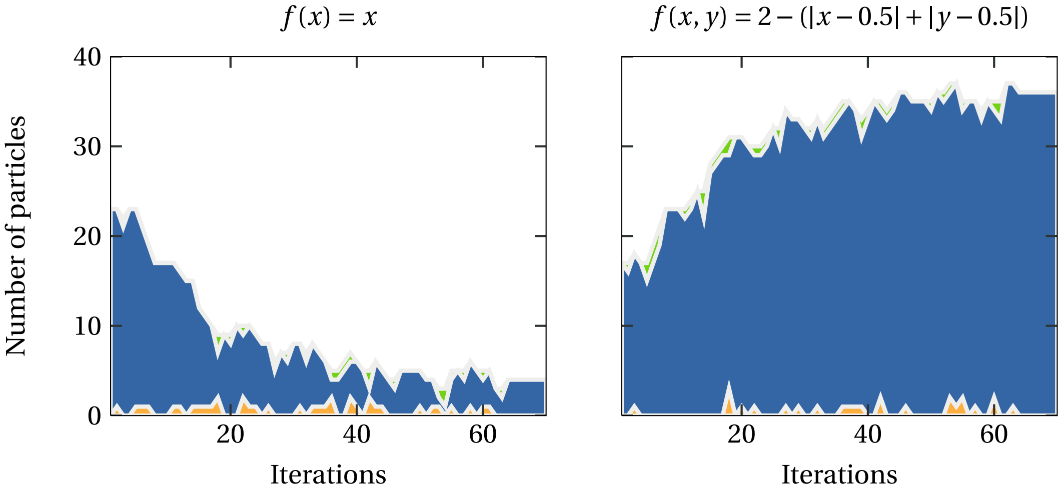


Figure 3.7 – Particle flow for subspace 1 (left) and subspace 2 (right). The green (upper) and orange (lower) areas show respectively the in- and out-flow of particles in each subspace.

iterations. The probabilities P_e, P_l and P_g were respectively 0.01, 0.05 and 0.05, i.e. constant which allows us to analyze their properties in a more straightforward manner than using iteration dependent probabilities as show in figure 3.5. All particles were initialized in the region $[0, 0.25]$ for both x and y to better show the effect of the particles converging on the maxima.

Figure 3.7 show the flow of particles between the two subspaces. The particles quickly converge on their respective maxima (not shown in the figure). For this simple problem, the population sizes of both subspaces can be easily calculated in the limit of the iteration using equation 3.6. Given that all particles will at some point have visited both subspaces (due to P_e), such that the global best and local best are both located in the second subspace. This results in a probability $P(s_1|s_2) = P_e$ and $P(s_2|s_1) = 1-(1-P_e)\cdot(1-P_l)\cdot(1-P_g)$. Given the probabilities as defined before, this results in $P(s_1|s_2) = 0.01$ and $P(s_2|s_1) \approx 0.1$. Thus the final populations would be approximately, on average as given in equation 3.8.

$$\begin{aligned} S1 &= P(s_1|s_2)\cdot\frac{N}{P(s_1|s_2)+P(s_2|s_1)} \approx 0.1\cdot\frac{40}{0.11} \approx 37 \\ S2 &= P(s_2|s_1)\cdot\frac{N}{P(s_1|s_2)+P(s_2|s_1)} \approx 0.01\cdot\frac{40}{0.11} \approx 3 \end{aligned} \tag{3.8}$$

Figure 3.7 show the trend towards these population sizes (though the simulation would have to be prolonged further to approach these values).

3.2.3 Applications

We briefly briefly discuss one previous application and one future application of MMPSO to show how this algorithm can be applied to a specific set of robotics problems.

Automatic gait generation in modular robots

In Pouya *et al.* (2010) we explored the generation of locomotion gait patterns for a modular robot named Roombots (Spröwitz *et al.*, 2010) using MMPSO. This work did not focus on the specifics of the optimization algorithm used, but rather on the control methodology of generating locomotion for modular robots. One module of this robot has 3 degrees of freedom (DOFs) (see Spröwitz *et al.* (2010) for details about the robot structure). One particular feature which makes Roombots an interesting platform for studying gait generation is that each DOF can continuously rotate, allowing a diverse array of locomotive behaviors. Two Roombots modules joined together are termed a *Metamodule*. The goal of this work was to explore locomotion modes of a Roombots *Metamodule*. The peculiar placement of the degrees of freedom of the *Metamodule* however make it hard to design locomotion controllers by hand.

If all 6 degrees of freedom of the *Metamodule* would have the same control law, then a standard PSO would have sufficed to optimize the various controller parameters. In this work however we were interested in exploring combinations of three different control modes for each of the DOFs: oscillation (i.e. sinusoidal), continuous rotation, and locked, in which the DOF is controlled to remain at a certain constant offset.

To explore combinations of these different control modes, we have successfully used MMPSO to select control modes for each of the DOFs. In MMPSO terminology, there were 6 (identical) pools (one for each DOF). Each pool consisted of three parameter groups (one for each control mode). The open control parameters to be optimized (for each DOF i) were the oscillation amplitude R_i, the oscillation or locked offset X_i and a phase bias ψ_{ij} controlling the phase relationship between neighboring DOFs. The MMPSO pool for each DOF i is given by: $[(R_i, X_i), (), (X_i)]$, with groups for respectively the oscillation, rotation and locked modes. Note that there are no parameters for the rotation mode and that the offset X_i is shared between the oscillation and rotation modes. In terms of MMPSO subspaces, there are a total of $3^6 = 729$ different subspaces to be explored. One possible MMPSO subspace is given in equation 3.9:

$$\{(R_1, X_1), (R_2, X_2), (X_3), (), (), (X_6)\} \tag{3.9}$$

where the two DOFs are oscillating, the third and last DOF are locked and the fourth and fifth DOF are rotating.

We ran MMPSO to optimize at the same time the control mode configuration and the control parameters. One of the main outcomes of that work shows that allowing optimization of so-called Hybrid control modes, selected by MMPSO, generally outperforms Pure control modes (such as only oscillatory or rotational modes for all the joints). The choice of the migration probabilities P_e, P_l and P_g gives precise control over how many iterations (on average) particles explore different subspaces and can be chosen informatively as it is straightforward to calculate how much time particles remain in the same subspace on average. For more details on this particular work, we refer to Pouya *et al.* (2010).

Co-design of mechanics and control of a wearable exoskeleton

MMPSO has been designed for applications where there are a certain (known) set of design choices to be made for (sub)parts of the system. This leads to a combinatory number of possible solutions to be explored. One interesting application of MMPSO for robotics is the co-design of the mechanics (or morphology) and control of a robot. In chapter 5, we use MMPSO for the co-design of the morphology and control of a wearable, non-anthropomorphic exoskeleton. Briefly, the main idea here is to first assume the human body to be a given, fixed mechanical structure. This “system” is then augmented with parallel structures composed of various components (linear/revolute actuators and rigid links), composing the wearable robot. Morphological parameters to be optimized are related to actuator placement and segment lengths. At the same time, open control parameters for controlling the actuators have to be optimized, which similarly to our work described before can have different control modes. The augmented system can then be evaluated on certain tasks such as locomotion assistance. MMPSO can be used here to explore the different combinations of mechanical parts to construct the exoskeleton attached in parallel to the human body, as well as the control of this exoskeleton, simultaneously.

3.2.4 Discussion

We have described a novel PSO based algorithm for optimizing specific optimization problems combining real-valued parameters with certain discrete choices in the type of solution being explored. Although the algorithm is suited only for these specific type of problems, we believe that it provides a valuable addition to the variety of existing modifications of the base PSO algorithm. The work explains in detail how principles of migration, inspired by genetic algorithms, can be applied to PSO in a collaborative way such that multiple, partially-overlapping parameter subsets can be explored simultaneously. The use of proper migration probabilities which separate *exploration*, *local exploitation* and *global exploitation* and their semantics makes choosing values for these probabilities well defined and understandable. The resulting behavior can be analyzed in terms of these probabilities and makes it easier to design the probability functions. Furthermore, the two-layer approach of the algorithm allows for any number of extensions of the base PSO algorithm to be used without any additional modifications to the outer layer algorithm.

This work was published in van den Kieboom *et al.* (2013).

3.3 Multi objective particle swarm optimization

Particle swarm optimization, like many other population-based optimization methods, is a single objective optimization method. Many optimization problems however have multiple objectives. The classic, and relatively easy way, to incorporate multiple objectives in such algorithms is to design an objective function which maps multiple objectives to a single objective, using a transfer function. This method is often called *aggregation* or *scalarization* (Coello, 1999). Effectively, the dimensionality of the objective space is reduced by projecting the objective functions onto a single dimension.

The problem with this type of approach should be obvious. By reducing the dimensionality through projection, an objective trade-off surface is created where combinations of different objective values result in the same, indistinguishable reduced objective value. Two commonly used objective aggregation functions are the weighted sum

$$f(\boldsymbol{s}) = \sum_i \alpha_i f_i(\boldsymbol{s}) \tag{3.10}$$

and weighted product

$$f(\boldsymbol{s}) = \prod_i f_i(\boldsymbol{s})^{\alpha_i} \tag{3.11}$$

, with f the final objective, $\boldsymbol{s}$ the problem solution, α_i a weighting constant and f_i the original objectives. Both of these projections allow for manipulation of the trade-off surface through weighting the individual objectives, however their objective gradient is not the same. Figure 3.8 provides an intuitive representation of the objective trade-off surface created by the two projections.

Note that the optimal solution does not change, yet the gradient does. Therefore, the path to get to the optimum has changed, and in a significant way. Following the gradient will result in faster convergence towards the true optimum when using a product aggregation. Of course, this is only clear for simple cases like the ones depicted above. In reality, objective functions are much more irregular, and choosing the right projection beforehand can be difficult.

3.3.1 Multi objective optimization

The aggregation method as just explained does not actually represent a truly multi-objective optimization. The reason is that even though aggregation methods create a trade-off surface, the trade-off itself cannot be observed because all values on it are equal. It is therefore still a global optimization (i.e. resulting in a single optimum found). On the other hand, multi objective optimization is interested in retrieving the trade-off surface itself.

A good overview of various adaptations of PSO for multi objective optimization problems is given in (Reyes-Sierra and Coello, 2006). For completeness, the most important approaches

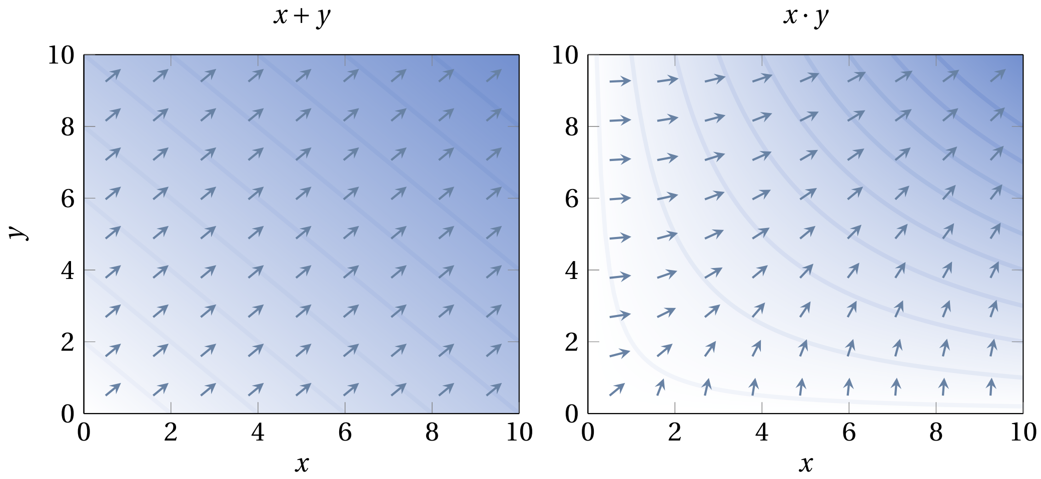


Figure 3.8 – Comparison of the effect of two projections of multiple objectives. On the left, a weighted sum aggregation leads to a linear trade-off surface between the two objectives. While on the right, a weighted product creates a non-linear trade-off surface. Lines on both plots indicate equivalent objective values while arrows indicate the gradient of the objective function.

are listed in table 3.1.

3.3.2 Multi objective PSO using lexicographic ordering

Lexicographic ordering optimization is another multi objective optimization method from the family of *a priori* optimization methods (Miettinen, 1999; Hu and Eberhart, 2002; Marler and Arora, 2004). As the name suggests, a priori methods need some a priori known information to reduce the set of Pareto optimal solutions in some way. The idea behind lexicographic ordering is to pre-assign a fixed ordering to the objective functions. Given this ordering, the objectives are optimized in sequence. Given a minimization problem, lexicographic ordering base optimization is formulated as:

$$\min f_i(\boldsymbol{x}) \qquad \text{such that} \tag{3.12}$$

$$f_j(\boldsymbol{x}) \leq f_j(\boldsymbol{x}^*), \qquad j = 0, \cdots, j = i-1 \tag{3.13}$$

where f_i is the i^{th} objective function, $\boldsymbol{x}$ is a solution vector and $f_j(\boldsymbol{x}^*)$ is the optimal solution of objective j. In other words, lexicographic ordering methods treat multiple objectives as inequality constraints, which are optimized in sequence. Note that the optimal solution $f_j(\boldsymbol{x}^*)$ has to be known for all but the last objective. If these optimal solutions are not known, then lexicographic ordering can not be used.

To apply this, the main idea is then to consider $N-1$ objectives to be objectives which can be formulated such that they are considered to be optimal within some range. For example, given an objective being locomotion at a desired speed, one can say instead that within

Table 3.1 – Multi objective PSO methods

Method	**Description**
Sub-population	In this approach, the population of particles is subdivided and each sub-swarm thus created, optimizes a single objective. The sub-swarms then share information about best found solutions and recombine their populations accordingly.
Pareto	Pareto based methods use the principle of Pareto optimality to explore the multi-objective trade-off surface. A solution to a multi objective function is said to be Pareto optimal if any improvement in the direction of any one objective would necessarily lead to a decrease in one or more of the other objectives. Furthermore, one solution is said to dominate another if it is better on every objective. The set of solutions that are Pareto optimal lie on the Pareto front and the goal is then to find this front. This is usually done based on the set of non-dominated solutions (i.e. the currently known set of solutions which are not dominated by any other solution).
Combined	If nothing else, the PSO research community has an excellent record of combining different methods to obtain new methods. Partly due to the fact that PSO is a very simple algorithm, and thus easy to extend, but also because many improvements to PSO are developed orthogonally. Recombining methods is a popular way to create new and more complicated methods which can be tailored to specific problem domains.

0.01 m/s of the desired speed, the difference is not significant. As soon as the objective of speed is reached, the next objective (for example energy efficiency) can be optimized instead. Multiple objectives can then be optimized in sequence until reaching the last objective which is minimized until a termination condition is reached. The ordering of the objectives can be significant, because it changes the search path. Whether or not this is a problem depends on the reachability of the next objective through the fitness landscape, having satisfied the previous objectives, and is problem specific.

Since particle swarm optimization is based solely on the ranking of solutions, and not on absolute fitness values, lexicographic ordering can be readily applied. The optimization can be designed by specifying N objective functions and $N-1$ objective constraints, which determine when an objective has been satisfied such that the next objective can be optimized in sequence. The N^{th} objective is minimized until a stopping criterion is met. Particles can then be ranked first based on the number of objectives they satisfy (in sequence) and then based on their fitness value of the current objective they are optimizing for.

As an example, let us look again at the example optimizatin of the Six-hump camel function as shown in figure 3.2. We can separate the function from equation 3.5 into contributions from x,

y and the cross of xy:

$$f_x(x,y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 \tag{3.14}$$

$$f_y(x,y) = (-4 + 4y^2)y \tag{3.15}$$

$$f_{xy}(x,y) = xy \tag{3.16}$$

$$f(x,y) = f_x + f_y + f_{xy} \tag{3.17}$$

Figure 3.9 shows the separated functions graphically. From these figures, it is straightforward to see where the minima of this function should lie (i.e. the sum of the three figures). The sum of the f_x and f_y functions creates two minima at the center top and center bottom. The effect of f_{xy} is then to skew these minima towards the top left and bottom right.

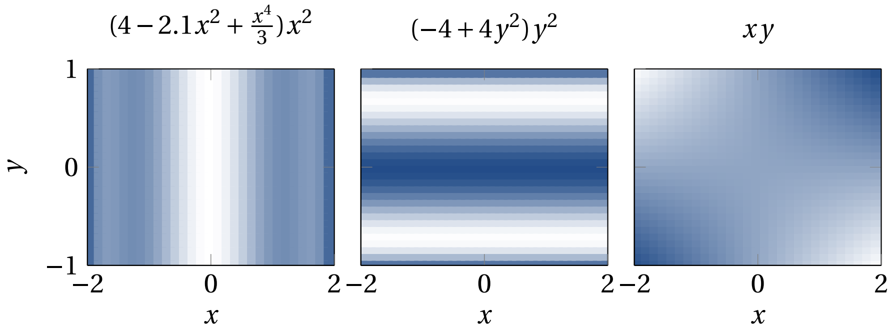


Figure 3.9 – Separation of the Six-hump camel function into contributions from (left to right) x, y and xy.

We can apply a lexicographic ordering method to the optimization problem by first optimizing for $f_x(x,y)$, then for $f_y(x,y)$ and finally for the full function $f(x,y)$, in sequence. We formulate the lexicographically ordered objectives by an objective and a condition until when the objective is to be optimized before optimizing the next objective. These objectives are listed in table 3.2. Note that the conditions are chosen somewhat arbitrarily in this example, but illustrate the main idea behind lexicographic ordering.

Table 3.2 – Lexicographic objectives of separated six-hump camel function

Objective	**Until**
1. $f_x(x,y)$	$f_x(x,y) < 0.5$
2. $f_y(x,y)$	$f_y(x,y) < -0.9$
3. $f(x,y)$	-

The results from running the lexicographic method based PSO are shown in figure 3.10. Here particles are seen to first optimize for f_x, i.e. traveling towards the x center of the parameter space, ignoring y. Then particles converge on the minima imposed by f_y while finally optimizing for the full function.

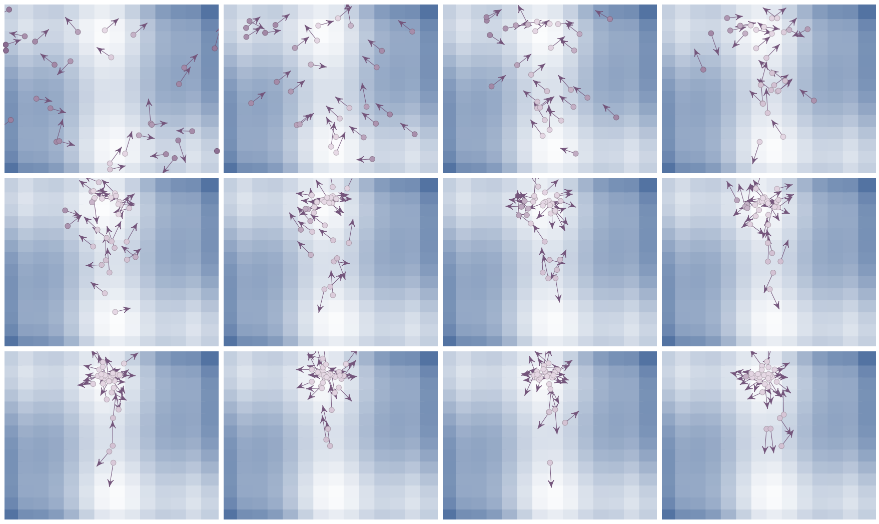


Figure 3.10 – PSO optimization with lexicographic ordering on the Six-hump camel function as defined in equations 3.14 to 3.17. The particles start out with random initial positions and velocities in the 2D parameter space. Particles then optimize for f_x, f_y and f in sequence and finally converge on one of the global minima.

Although lexicographic ordering is argued to work well only when considering a small number of objectives (Coello, 1999), it lends itself naturally to optimization problems treated as sequential learning of multiple objectives. Consider the example of optimizing a gait for a locomotion task. There are several objectives involved in successfully completing such a task, keeping balance as to not fall over, keeping enough ground clearance, gaining enough speed and minimizing for energy expenditure. Naturally speaking, it would be hard and unpractical to try to obtain all objectives at the same time. It is more important for example not to fall over first, and only then to try to optimize your energy expenditure.

Finally, lexicographic ordering can be seen as a special treatment of Pareto optimality. Indeed, since objectives are formulated as inequality constraints, improvements on a particular objective can only be made within the constraint space of the previously obtained objectives. Furthermore, within the objective constraint space, objective differences are deemed insignificant. Therefore, in terms of Pareto optimality, solutions are always Pareto optimal with respect to previous objectives. The difference is of course that whereas a Pareto based method will try to explore the full Pareto front of multiple objectives, a lexicographic method does not because objectives are only considered sequentially. In other words, it does not explore the objective constraint space, other than to optimize for the next objective. It is therefore still a global optimization method.

3.4 Large scale population-based optimization

Population-based optimization methods are computationally expensive. They rely on the evaluation of large populations to effectively explore a problem's search space. As the number of parameters to be optimized grows, so must the population size. Additionally, depending on the type of problem, it can take many iterations before the population converges to a (locally) optimal solution. Finally, because most population-based optimization methods are stochastic in nature (e.g. initial random population, random mutation operations, etc.), optimizations need to be repeated to obtain statistically meaningful results.

During the course of this thesis, an open source framework was developed which makes it easy to do these type of large scale optimizations on consumer grade, off the shelf hardware. Apart from the fact that to the best of our knowledge, there currently do not exist good alternatives to perform these type of optimizations in a managed, multi-user environment, there is little novelty nor scientific value (in terms of research) in the design or development of this type of framework. It is however a fundamental tool without which the presented research (and others) could not have been done. Since it has been written for general purpose use, and distributed freely, it provides a valuable scientific research tool.

3.4.1 Conceptual overview

Figure 3.11 shows a schematic overview of the optimization framework. There are three distinct layers (user, server and workstation) in the system. The *user* layer runs the actual optimization process which will produce tasks that have to be evaluated. These tasks are passed to the *server* layer which queues them in a task queue. The server acts as a central hub to distribute any tasks it receives to the available workstations. A *workstation* in turn receives a single task from the server and executes it. When the task has been evaluated, the result is sent back to the user through the server layer. The different layers can run on a single PC, but can also be placed on different networked PCs.

The overall concept of the framework is such that no specific restrictions are placed on the type of optimization algorithm or the manner in which a task has to be evaluated. Therefore, the framework can easily be used for many different and concurrent tasks.

3.4.2 User layer

The user layer represents the front-end layer which is run by a user of the system. This layer is responsible for running the optimization algorithm which produces tasks to be evaluated. As can be seen in figure 3.11, each optimization, run at the user layer, is encapsulated in a *Job* process. The job drives the optimization, sends tasks to the server layer, and feeds retrieved results back into the optimizer.

The optimizer consists of a population (of tasks) that need to be executed, an optional function

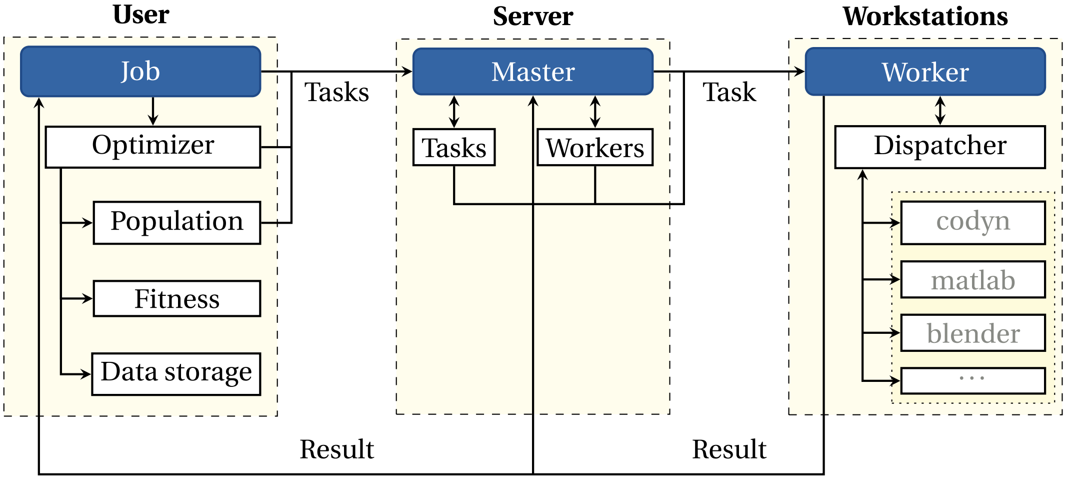


Figure 3.11 – Schematic overview of the optimization framework architecture

which combines a multi-objective fitness evaluation into a single fitness value, and a data storage to store the results of the optimization. The optimizer population is a set of tasks that can be executed independently. For example, in genetic algorithms this would be a single generation of individuals. These represent the set of tasks that can be distributed at the server layer. The fitness function represents a mathematical expression that can be used to transform multiple objectives into a single fitness value, used by the optimizer.

A description of the task that is sent to the server layer is given in table 3.3. Once a task is executed, a result message as described in table 3.4 is sent back to the user layer.

Table 3.3 – Task Message

Name	**Description**
Id	A unique task identifier
Dispatcher	The dispatcher with which to evaluate the ask
Parameters	A $\{name \rightarrow value\}$ dictionary of parameters to be evaluated
Settings	A $\{name \rightarrow value\}$ dictionary of settings to be passed to the dispatcher

Table 3.4 – Result Message

Name	**Description**
Id	The unique task identifier
Status	Whether the execution was successful or not
Fitness	A $\{name \rightarrow value\}$ dictionary of fitness values
Data	A $\{name \rightarrow value\}$ dictionary of additional, custom data to be stored with this task/solution

Job specification

Jobs to be executed on the framework are specified in a simple XML format. It specifies which optimization algorithm to use, specific parameters for this algorithm (if any), the problem parameters to be optimized, the fitness function to use and which dispatcher should handle evaluation of the task. An example of a job specification is given below:

```xml
<?xml version="1.0" encoding="utf-8"?>

<job name="example">
  <optimizer name="pso">
    <!-- settings specific to the type of optimization algorithm -->
    <setting name="population-size">40</setting>
    <setting name="max-iterations">200</setting>
    <setting name="max-velocity">0.6</setting>
    <boundaries>
      <boundary name="speed" min="100" max="1000"/>
    </boundaries>
    <parameters>
      <parameter name="left" boundary="speed"/>
      <parameter name="right" boundary="speed"/>
    </parameters>
    <fitness>
      <!-- combine two fitness values in a single fitness -->
      <expression>radius - from_origin</expression>
    </fitness>
  </optimizer>
  <dispatcher name="codyn">
    <!-- settings specific to each dispatcher -->
    <setting name="world">$OPTIMIZATION_JOB_PATH/../simulator/impedance</setting>
    <setting name="max-time">10</setting>
  </dispatcher>
</job>
```

3.4.3 Server layer

The server layer consists of a single process which acts as a distribution center for tasks to be evaluated. This layer is used to allow multiple optimization processes to be run, while sharing the workstation resources that are currently available. The tasks are scheduled fairly with respect to the estimated execution time and a task priority. The server automatically discovers new workstations as they come online through a simple discovery protocol. When new tasks are received from the user layer, it schedules these tasks (according to their priority) onto a task queue. Whenever a workstation becomes available, the task is sent to this workstation to be executed. The workstation then sends the result back to the server, which in turn relays it back to the user layer.

3.4.4 Workstation layer

The workstation layer is responsible for executing a single task, and sending the result back to the server layer. Each task is received from the server layer with the task information as specified in table 3.3. The worker process then resolves a dispatcher process from the task description that is to be used to evaluate the task. When the correct dispatcher is located, this dispatcher will be executed with the task that has to be evaluated. From the dispatcher, the worker will receive a result in terms of fitness, which it then relays back to the server layer.

Dispatcher

Up to the dispatcher, the framework is entirely task agnostic. The dispatcher is responsible for actually evaluating a task and there are several dispatchers provided in the framework. Additionally, it is also easy to write custom dispatchers, for example for other simulators. There are C++, Python and C# API's available from which new dispatchers can be easily constructed. A dispatcher is a simple standalone process which receives a task description on its standard input and writes back a response on its standard output. Dispatcher management is entirely handled by the workstation layer.

3.4.5 Results and analysis

All information related to running a task (i.e. all information in the job specification) and all intermediate results of an optimization run are stored in a SQLite database. This is a convenient way to store data in a structured way and SQLite is a widely supported format. Storing all intermediate results leads to relatively large databases, but it allows for exact reconstruction of the entire optimization process. This also means that running jobs can be suspended and resumed, even in case of failure (for example accidentally closing the user process, or a crash). Special care is taken to store the state of the random number generator so that resuming a job at a later time does not cause differences in any way.

The obtained results database can be inspected using a graphical user interface providing information on the fitness progression and obtained best solutions. Furthermore, the data can be exported to be analyzed in Matlab using a provided toolbox.

3.4.6 Availability

The optimization framework software is made freely available under the GPL license. It is made available at http://optimization.codyn.net/, including sources, documentation and binary packages.

3.4.7 In the wild

The tools, simulators and optimization frameworks that have been developed in the course of this part I have enabled, outside of the work presented in this thesis, a number of other works of research. Both cȯdyn, as well as the framework for large scale optimization have provided opportunities which would have otherwise been more difficult and importantly, more time consuming. Of course, the importance of having such software available should not be overstated, it is certainly not *essential*. On the other hand, it does provide a set of very convenient research tools which can be quickly applied. Here we briefly present works to which both frameworks have contributed.

Roombots

In Spröwitz *et al.* (2010); Pouya *et al.* (2010); Moeckel *et al.* (2013), we used cȯdyn to implement a central pattern generator based controller for a modular robot called Roombots (Sproewitz *et al.*, 2009). Here the large scale optimization framework was used to optimize various locomotion controllers. By optimizing the controllers, gaits which utilized the unique, and unintuitive to control, degrees of freedom of the Roombots to move around in ways that were hard to engineer.

Co-evolution of morphology and control of virtual legged robots for a steering task

In Larpin *et al.* (2011) we use co-evolutionary strategies to optimize for the control and the morphology of legged robots, quadrupeds in particular. Here we evolved task specific morphology and looked at how body structure influences the steering capabilities of relatively simple robotic structures. At the same time, we used cȯdyn to easily create structured networks of coupled oscillators for which parameters were optimized.

Model-based and model-free approaches for postural control of a compliant humanoid robot using optical flow

Through the use of particle swarm optimization to train neural networks, postural control of the CoMan robot using optical flow could be realized by learning a mapping from sensor information to the adaptation of central pattern generators. Here cȯdyn was used for the implementation of adaptive frequency oscillators (Buchli *et al.*, 2005) and the optimization software was used to train the neural network.

3.5 Conclusion

In this chapter we have presented a general, but brief, introduction into optimization methods with a particular focus on population-based optimization methods. Population-based methods make use of nature inspired algorithms where competition and cooperation play important roles in the search for solutions to a problem. They are suitable for the open-ended exploration of large search spaces and require due to their meta-heuristic nature, very little knowledge of the problem domain.

Among many population-based methods, Particle Swarm Optimization is a relatively recent method loosely inspired by swarming behavior of biological organisms. As a simple algorithm, with very few parameters, it shows the capability of exploring search spaces, using cooperative strategies.

We are interested in applying Particle Swarm Optimization to problems with variable parameter configuration spaces, where a fixed number of possible solution structures are known, each with a corresponding (and possibly overlapping) parameter set. Examples of such problems include choosing between various actuator schemes (each scheme with its own set of parameters) or the co-design of the morphology and control of a robotic structure, where the possible desired set of structures are known.

To this end, we have presented our novel Metamorphic Particle Swarm Optimization method which combines Particle Swarm Optimization with mutation operators inspired by genetic algortithms to transfer particles between different parameter (sub)spaces. By applying cooperative strategies to the parameter subspace tranfer probabilities, we show how the principles from Particle Swarm Optimization for continuous parameters can be used also for the optimization of discrete sets of parameter configuration spaces.

Finally, we briefly presented our open and freely available framework for performing large-scale, population-based optimizations in a multi-user setting. The framework does not necessarily present a novel approach to these type of optimizations, but has been instrumental in several bodies of research. The value of the contribution is in the availability and ease of use, allowing for rapid replication of results presented in this thesis. Due to its problem, task, and task evaluation agnosticity, it can be generally applied beyond the work presented here and can be a useful tool for anyone interested in doing large-scale optimizations.

Human locomotion and assistance Part II

4 Optimization of natural human gait

When it comes to research on biped locomotion in robotics, there exist several distinct research directions. We briefly discuss several of the major approaches. When we look at the energetics of bipedal locomotion, one area of research that has been fundamental is that of passive dynamic walking (McGeer, 1990). Despite this works relative age in the field of robotics, it has given key insights into minimal energy solutions for biped locomotion by maximally exploiting the natural dynamics of the mechanical system. Passive dynamic walkers often exhibit a remarkable similarity to human locomotion (Collins *et al.*, 2005), suggesting that humans make use of similar concepts. Since its original inception, advances have been made to increase stability and controllability (Wisse, 2005) and to move towards walking in 3D (Collins *et al.*, 2001; Wisse *et al.*, 2001).

On the other end of the spectrum, one could argue, is biped locomotion by means of trajectory/path planning. This often involves (indirect) control of the Zero Moment Point (ZMP) (Vukobratović and Borovac, 2004; Vukobratovic and Juricic, 1969). Motions are planned such that the ZMP, the point on the ground at which there is no net moment, remains within the robot's foot support area. This guarantees dynamic stability of the robot. The Honda humanoid robot (ASIMO) is particularly well known to use ZMP trajectory planning (Hirai *et al.*, 1998; Hirose and Ogawa, 2007). More recently Huang *et al.* (2001) have shown that it is not necessary to plan the ZMP trajectory in full. Instead one can use an iterative approach. In Kajita *et al.* (2003), ZMP based control is combined with inverted pendulum based approaches, attracting the robot towards a future ZMP reference. Although trajectory based approaches to biped walking have been very successful, their resulting control is often stiff (position based) and lacks the agility and dynamic of human walking.

Moving away from engineering solutions to biped locomotion, researchers have increasingly been looking at biologically inspired walking. These works use biologically inspired artificial muscle models (based on Hill-type muscle models of varying complexity) to model the major human muscle groups. It has been shown that minimization of metabolic muscle energy is a sufficient objective to obtain human like walking performance in these types of models (Anderson and Pandy, 2001). Furthermore, using relatively simple muscle reflexes, natural biped walking can be obtained in simulation (Geyer and Herr, 2010; Wang *et al.*, 2012). These works indicate that the biomechanical system is particularly well “designed” for locomotion.

One particularly important aspect of biomechanical systems that lead to its performance, robustness and agility, is the ability to task-dependently change the impedance of the system (Buchli *et al.*, 2010). Variable impedance control has been used in human-robot cooperative manipulation tasks, where it is important to be able to “give in” to the human collaborator (Ikeura *et al.*, 2002; Rahman *et al.*, 2002; Duchaine and Gosselin, 2007). It has also been shown that low impedance control can be beneficial to design robots that locomote both safely and robustly. In Park (2001) the concept of variable impedance is applied to biped locomotion. Joint trajectories are first generated using a gravity-compensated inverted pendulum model (Park and Kim, 1998). The impedance between the feet and the ground are then modulated to effectively moderate impact forces at ground contact and is shown to help in stabilizing foot placement.

In this chapter we investigate the role of joint-level impedance control for humanoid locomotion. The use of impedance control, as opposed to position control, is inspired largely by passive dynamic walkers. By allowing impedance to vary we expect that it is possible for optimization to exploit the natural dynamics (i.e. low impedance) more easily. In particular, we will look at the emergence of human like gaits by optimizing first principle objectives, i.e. obtaining a characteristic human like gait without explicitly optimizing for it. Our first study will develop a method for the simulation and optimization (using the tools developed in the previous chapters) of impedance controllers to obtain a minimal model explaining various global human gait characteristics. Having obtained this minimal model, the same methods are applied to optimize for human like gaits on a model of a humanoid robot. We look at the differences in obtained gaits and see the importance of (bio)mechanics versus a mimetic approach for humanoid robot design. Finally, we design a simulated perturbation study to investigate the emergence of the role of variable impedance towards disturbance rejection.

4.1 Human gait optimization

We begin by developing our method for the optimization of a minimal model for human gait optimization and the role of joint-level variable impedance control. This study originated from the following hypotheses

1. Human like gaits arise spontaneously when using impedance control and optimizing only for mechanical energy expenditure.
2. Gait quality (kinematic similarity, global human gait characteristics) *increases* with increasing modulation of impedance.
3. Energy expenditure *decreases* with increasing modulation of impedance.

We test these hypotheses using simulations of a simple humanoid model in the sagittal plane while optimizing variable impedance controllers for each of the joints using Particle Swarm Optimization. The remainder of the section is organized as follows. First we describe in section 4.1.1 the simulation model and control. We then continue to describe the optimization method in section 4.1.2. Finally we describe the main results in section 4.1.3 where we confirm our first hypothesis fully and show trends towards the second and third hypotheses. We conclude with a discussion in section 4.1.4.

4.1.1 Model

The simulated biped is modeled after an adult sized human, using kinematic and inertial properties derived from Winter (2009). Figure 4.1 lists these quantities as used in this work. Note that the segment length l and center of mass quantities CoM_x and CoM_y are specified proportional to the total model height. The center of mass quantities are proportional to the segment length l and the mass m is specified proportional to the total model mass. The model height and mass used are respectively 1.80 m and 70 kg in this work.

Joint	CoM_x	CoM_y	l	m	r_g
torso	0.000	0.030	$n.a$	0.678	0.90
up leg	0.000	−0.108	0.25	0.100	0.32
low leg	0.000	−0.108	0.25	0.047	0.30
ankle	0.004	−0.002	0.04	0.011	0.48
toe	0.001	0.000	0.03	0.003	0.10

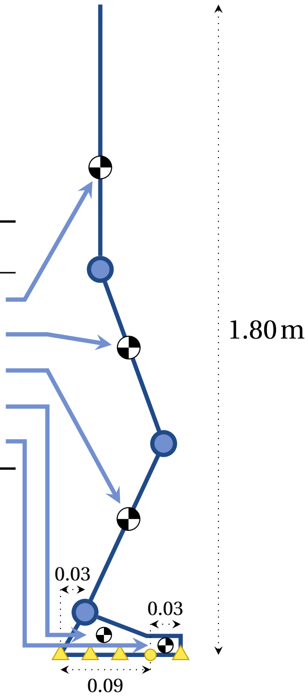


Figure 4.1 – Kinematic and inertial properties of the biped model. Kinematic quantities are proportional to the total model height and the mass is given proportional to the total model mass. r_g indicates the segment radius of gyration, from which the inertia can be derived. The schematic on the right shows approximate center of mass locations and contact point locations on the foot (triangles).

We focus only on walking in the sagittal plane. Including the floating base, our model thus has a total of 11 degrees of freedom. We only actuate the hip, knee and ankle joints of this model. The toe joint is modeled using a critically damped spring with a spring constant of $12\,\mathrm{N\,m\,rad^{-1}}$. The only biomechanical effect that we simulate is that of ligaments at the knee to prevent hyper extension, by means of an exponential spring/damper ($\tau_l = Kl_p \cdot (q_l - q)^3 - Kl_d \cdot \dot{q}$, with $Kl_p = 1e6$, $Kl_d = 1e2$ and $q_l = -2°$). This facilitates both the exploitation of passive force and eases the optimization's ability to find initial solutions.

The rigid body dynamics of this model is modeled and simulated using cȯdγn. For the floating base, we use a special planar joint (available in the cȯdγn standard library) allowing rotation on the Y axis, and translation in the X and Z direction. The human model is parametrized such that we can easily change the total length and have all parameters automatically derived using scaling laws obtained from Winter (2009). The contact model used for this particular study is the soft contact model of cȯdγn, i.e. a spring/damper contact model including basic coulomb friction. Each foot has 5 equally distributed points on the bottom surface at which contact forces are generated as soon as they penetrate the ground.

Controller

Each of the hip, knee and ankle joints are controlled using a simple variable impedance control law. This control has the following form:

$$\tau_i(t) = k_i(t)(\bar{q}_i(t) - q_i(t)) - b_i(t)\dot{q}_i, \tag{4.1}$$

where τ_i is the commanded torque on joint i, $k_i(t)$ is a time varying stiffness pattern, $\bar{q}_i(t)$ is a time varying desired joint angle pattern, $q_i(t)$ is the actual (measured) joint angle and $b_i(t)$ is a time varying damping pattern. Each of the control signals for the desired joint angle $\bar{q}_i(t)$, joint stiffness $k_i(t)$ and joint damping $b_i(t)$ are time varying, periodic signals that need to be specified to obtain the final control torque τ_i at each time. Further references to these variables will omit the explicit reference to time for brevity.

The actual control signals can be implemented in different ways, ranging from simple constant values to highly complex signals. In this work we are interested in exploring the influence of variable impedance control with respect to constant impedance control. Here we explore three levels of increasing control complexity as defined in Table 4.1, 1) a constant value, 2) a step function with controllable transition timing and 3) a piecewise polynomial function. Note that t is normalized on $[0,1]$ for one period of the signal. For the piecewise polynomial function we chose the piecewise monotone cubic ($N = 4$) Hermite spline (Fritsch and Carlson, 1980) due to its ease of construction and monotonicity property (i.e. it does not overshoot). We used 4 data points to interpolate the control signal. Furthermore, we ensure the resulting signal to be continuous and periodic by constraining $\dot{f}(0) = \dot{f}(1)$.

For the reference trajectory $\bar{q}$ we use the *ppoly* control type in all scenarios, using 4 data points. Figure 4.2 shows nominal joint angles during normal walking for the hip, knee and ankle. The piecewise interpolated trajectories show that 4 data points are sufficient ($R_{\text{hip}} =$

Table 4.1 – Control modes for k (same for b)

Type	Function	Parameters
1. constant	$f(t) = k$	k
2. step	$f(t) = \left\{ \begin{array}{ll} k_1, & \tau_1 < t < \tau_2 \\ k_2, & \text{else} \end{array} \right\}$	$k_{1,2}, \tau_{1,2}$
3. ppoly	$f(t) = \sum_n^N {}^p\alpha_n t^n,\ {}^p\tau_{(n-1)} < t < {}^p\tau_n$	${}^p\alpha_n, {}^p\tau_n,$ $n \in [0, N]$ $p \in [0, K]$

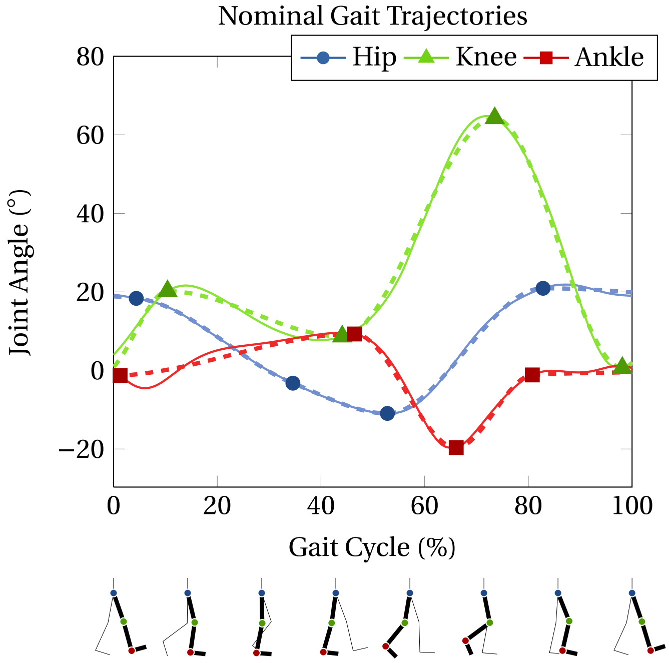


Figure 4.2 – Nominal joint angle trajectories, taken from Winter (2009). The dashed lines show the result of piecewise interpolating the data points, resulting in a sufficient representation of the nominal joint angle trajectories.

0.999, $R_{\text{knee}} = 0.998$, $R_{\text{ankle}} = 0.988$) to represent the nominal trajectories and preserve its primary characteristics.

Stability

It is possible to find solutions which have a stable *steady state* gait cycle (i.e. that do not fall over) only by means of the joint impedance control laws. Getting into this steady state however is not trivial without explicitly taking care of the gait initiation (on which we do not focus in this work) or adding active balance controllers. Instead we chose to add a simple external *assistive* torque on the floating base rotation degree of freedom during the initial phases of optimization: $\tau_a = Ka_p \cdot (\pm q_a - q) - Ka_d \dot{q}$, with $Ka_p = 3000$, $Ka_d = 20$ and $q_a = 5°$. This simple assistance is sufficient to allow the optimization to find initial walking solutions. Note that there is a neutral zone of $\pm q_a$ in which no assistive torque is provided. Also, the final solutions do not require the assistive torque except during initiation since they are optimized to stay within the neutral zone.

4.1.2 Optimization

The optimization algorithm used to optimize the control parameters is Particle Swarm Optimization using lexicographic ordering to optimize for multiple objectives in sequence (see section 3.3.2 for more details). Note that this is a standard PSO and not the previously developed MMPSO since we do not need to optimize the structural space. The constriction factor K and constants ϕ_1, ϕ_2 were set to $K = 0.729$, $\phi_1 = \phi_2 = 2.05$ as to ensure convergence of the swarm (see Clerc and Kennedy (2002)). The maximum velocity of the particles is limited to 0.6 (normalized to the parameter boundaries) and particles are set to bounce off of parameter boundaries (by means of reflecting their velocity vectors).

Objectives

We have three main objectives for the optimization: 1) we want to walk forward at a specified, constant speed, 2) we want to have a stable walking gait without the need for external assistive torque, and 3) we want to minimize the overall energy expenditure. We focus on walking at a specific, desired speed to restrict our method to exploring walking at a speed which should naturally lead to energy efficient locomotion. For 3) we choose torque being the simplest, first approximation of energy expenditure. In our approach, we do not explicitly optimize for other global gait qualities such as kinematic similarity to human data, since we hypothesize that human gait kinematics are a natural product of minimization for energy. However, to avoid exploitation of certain simulation inaccuracies (such as the contact model approximation) we add one additional criterion which specifies that at least one foot has to be in contact with the ground at any given time. This prevents exploiting the spring of the contact model to store and release energy, making the model jump from the ground.

To use lexicographic ordering, we formulate the just stated objectives as sequences of objective functions and constraints, as explained in section 3.3.2.

Table 4.2 lists the sequence of objectives stages that we used (objectives are *maximized*). The first stage ensures that solutions are simulated for the maximum amount of time without falling over. The second and third stage ensure that the solution reaches the desired target speed, with a maximum defined standard deviation of the measured speed over the whole gait. Speed match is the measured speed as a fraction of the target speed. We thus require speed to be within 95% of the desired target speed. We chose a walking frequency and speed based on data from Winter (2009) for normal walking. For all experiments, we set the desired

Table 4.2 – Lexicographic Objectives

Objective	Until		
1. time	time	=	max time
2. speed match	speed match	≥	0.95
3. -std speed	std speed	≤	0.1
4. -assist time	assist time	=	0
5. -non contact time	non contact time	=	0
6. -torque	∞		

forward locomotion speed to 1.3 m s^{-1} (or 4.7 km h^{-1}) and the walking frequency to 0.9 Hz. The fourth and fifth stages optimize for walking without assistance while having at least one foot in contact with the ground at all times. Finally, the last objective minimizes control torque as long as all conditions 1-5 have been fulfilled. Since we do not take special care of gait initialization, the initial seconds of the simulation can exhibit non-steady state behavior. We therefore ignore data from the first half of the evaluation period for all measured objectives, except time.

Note that as mentioned in section 3.3.2, the ordering of the various objectives can influence the optimization process. Although we did not extensively explore the effect of ordering, we emperically verified that it is important for the optimization to first optimize time and only then match speed. The reason is that this ensures that average speed is always measured over the whole simulation time (i.e. the model does not fall over). Although this is not the only manner in which this behavior can be achieved (for example, one could optimize explicitly for distance), the chosen objectives are simple and prevent exploitation of specific simulation artifacts.

Parameters

We use a differential encoding δx for the x position of the *ppoly* data points, such that $x_i = \sum_j^i \delta x_j / \sum_j^{N+1} \delta x_j$. Note that we need $N+1$ parameters for δx to satisfy the constraint that $\sum_i^N \delta x_i = 1$.

Table 4.3 – Optimizing Parameters

	constant	step	ppoly
angle	$p^N_{h,k,a}$		
stiffness	$k_{h,k,a}$	$k^{1,2}_{h,k,a}$, $t_{1,2}$	$k^N_{h,k,a}$
damping	$b_{h,k,a}$	$b^{1,2}_{h,k,a}$	$b^N_{h,k,a}$
number $(N=4)$	33	45	81

Table 4.3 lists the parameters to be optimized for each of the three different control modes. The subscripts h, k and a refer to respectively the hip, knee and ankle joint. We have adopted

the superscript ^{N} notation to indicate the parameters required for piecewise interpolation of N data points, which resolves to $N+1$ parameters for δx and N parameters for k, b or p (thus a total of $2N+1$ parameters are required). For the *constant* control mode, we need one stiffness and one damping parameter for each joint. The *step* mode needs one additional stiffness and damping parameter for each joint, as well as two parameters $t_{1,2}$ determining when the step occurs in the gait cycle. Finally, for the *ppoly* control mode we need the same parameters for stiffness and damping as we have for the joint angle. The bottom row of the table shows the total number of parameters being optimized for $N = 4$.

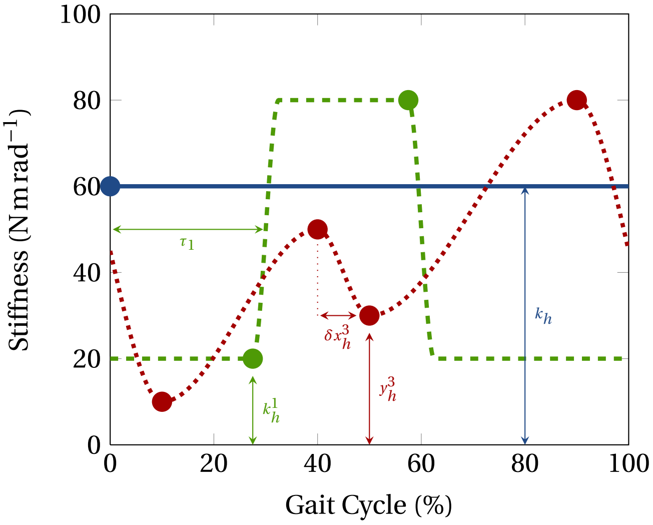


Figure 4.3 – Example hip stiffness signals resulting from the *const* (blue solid), *step* (green dashed) and *ppoly* (red dotted) control modes. The large dots represent the control points which need to be optimized to obtain the resulting signals. We only annotated one pair of parameters per signal in the figure for clarity (e.g. t_2 and k_h^2 are not shown). Note that we need $N+1$ parameters for δx and N for y (here shown for $N = 4$) for the *ppoly* signal.

Figure 4.3 shows example signals for each of the three control modes. Note that the more complex encodings also include all signals that can be obtained from the less complex encodings (i.e. the encoding for *step* can generate all *const* signals).

Stiffness values are bound in [0, 12000]$\mathrm{N\,m\,rad^{-1}}$ and damping values in [0, 60] $\mathrm{N\,m\,s\,rad^{-1}}$. The upper limits were determined empirically resulting in a very stiff controller. Joint angle parameter values are bound in [-0.8, 0.8], [-0.1, 1.5] and [-0.6, 0.6]rad for respectively the hip, knee and ankle joints. These values conservatively contain nominal joint angle trajectories.

Experiments

We run the PSO with 120 particles and for 500 iterations. Note that we use a relatively large number of particles since 1) we have a large number of parameters to optimize and 2) it is difficult to find initial solutions that manage to walk a few steps (which can then be refined).

We have thus used conservative numbers for particles and iterations which could possibly be reduced. Note that the proposed optimization strategy is used purely for offline evaluation of control laws and not as an online optimization procedure. For each control condition, we run the PSO 10 times with a randomized initial population. This results in a total of 30 runs of PSO, and a total of 1.8 million evaluations. We run this optimization on a cluster with 127 cores (11 dual quad core Xeon E5504, 2 GHz and 7 dual hexa core Xeon E5-2430, 2.2 GHz), using the large scale optimization framework developed in section 3.4. A single run took approximately one hour of real time.

4.1.3 Results

On average, the objectives as listed in 4.2 take respectively 2, 7, 13, 71 and 87 iterations for the first solution to be found that matches its constraints. Furthermore, on average 360 iterations out of the 500 iterations are spent optimizing for torque. During this time, the torque is reduced between 3 to 4 times.

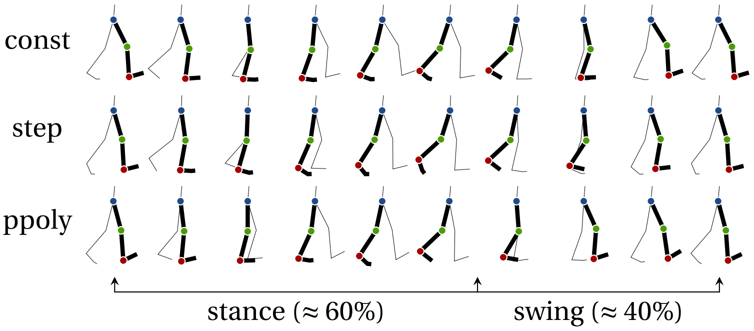


Figure 4.4 – Snapshots of one gait cycle of the best obtained solutions from optimization for each control mode. The gait is shown from heel-strike to heel-strike. All three gaits show similar characteristics and look qualitatively human like. The *ppoly* gait shows full knee extension, while the other two control modes show the knee slightly flexed. The torso leans slightly forward in all cases.

Figure 4.4 shows snapshots of one gait cycle for the best solution, in terms of torque, of each control mode. Even though there was no explicit objective to optimize for specific gait qualities, several global human gait characteristics can be observed. All gaits feature heel-strike, foot-roll, heel-rise, toe push-off and a double support phase. We also obtain a stance duration of ≈ 60% of the total gait duration, which is the same as the stance duration in normal human walking. Another characteristic of human gait is the occurring double peak in ground reaction force, caused by heel-strike and then toe-off. We observe a similar double peak in ground reaction force in our simulation results (data not shown).

Table 4.4 shows the obtained average and best obtained results for the three control modes. In all cases, the optimization reached the last stage, minimizing for torque. In addition to the minimum torque, Table 4.4 shows the correlations c_h, c_k and $c_a$between the hip, knee and

Table 4.4 – Average Performance

	Average				Best			
	$\bar{\tau}(\sigma)$	$\bar{c}_h(\sigma)$	$\bar{c}_k(\sigma)$	$\bar{c}_a(\sigma)$	τ	c_h	c_k	c_a
const	232.3 (44.3)	0.94 (0.07)	0.68 (0.30)	0.17 (0.50)	161.6	0.98	0.68	0.17
step	190.9 (53.3)	0.85 (0.07)	0.58 (0.35)	0.26 (0.22)	108.6	0.96	0.90	0.54
ppoly	216.3 (56.1)	0.93 (0.05)	0.76 (0.13)	0.21 (0.27)	146.7	0.89	0.69	0.13

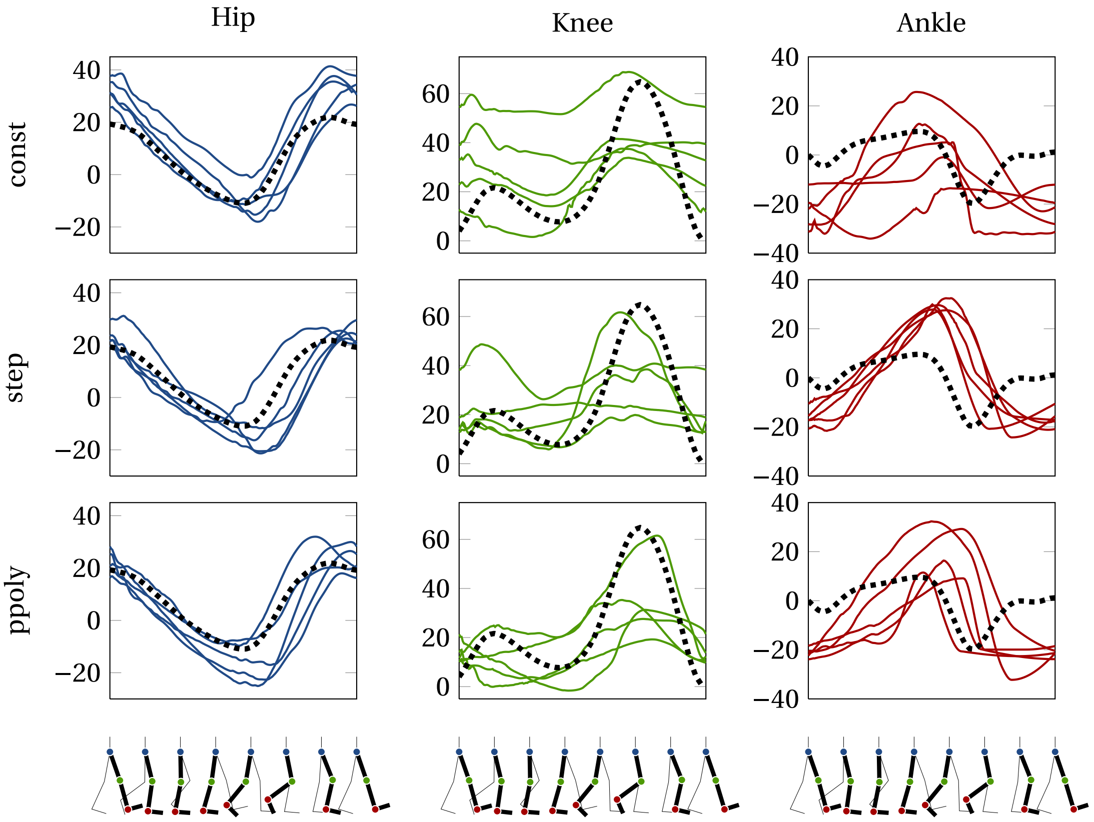


Figure 4.5 – Obtained joint angle kinematics after optimization. All angles are shown in degrees. The dotted lines are average human joint kinematics for normal walking (Winter, 2009). The solid lines are resulting joint kinematics after optimization, for the 5 best obtained solutions. Each row shows one of the *const*, *step* and *ppoly* control modes. The hip kinematics strongly correspond to normal human walking. For the knee, best results in terms of matching kinematics are obtained for the *ppoly* control mode. Ankle kinematics on the other hand are more consistent for the *step* control mode.

ankle joint angle kinematics and average human kinematics. The resulting kinematics are shown in figure 4.5. The kinematics of the hip are consistently close to normal human hip kinematics for all three control modes. The knee and in particular the ankle kinematics, on the other hand, do not resemble human kinematics to the same extent on average. For the knee, we generally find less flexion during swing and we do not always obtain the extension →

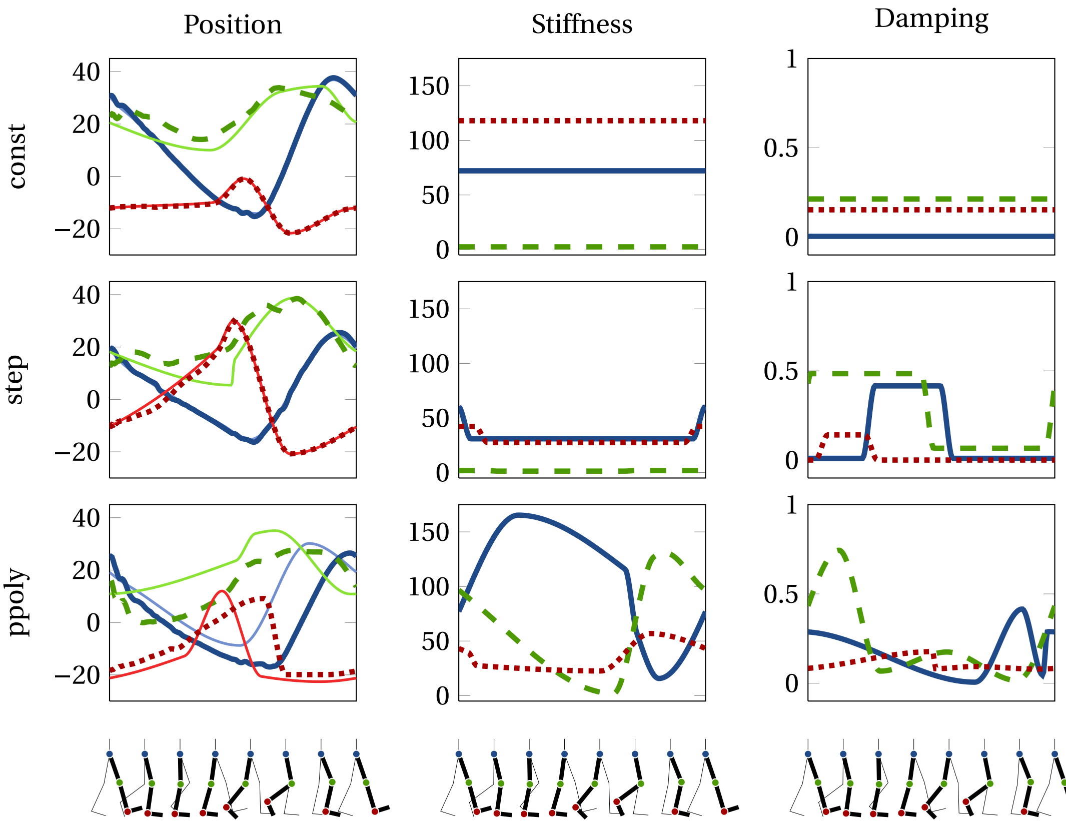


Figure 4.6 – Optimized joint control signals. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$. The *const* control mode stiffness is high for both the knee and hip joint, resulting in a close match between reference and actual joint angles. The *step* control mode features relatively low stiffness patterns, while still providing close tracking of the reference joint angle. Both the hip and knee joint tend to stiffen during heel-strike, presumably to stabilize ground impact.

flexion → extension during early stance. The ankle joint consistently shows more dorsiflexion as well as a higher peak plantarflexion. Furthermore, it tends to dorsiflex later during swing than normally observed in human walking. Whereas in normal human walking, the knee flexion during swing provides sufficient ground clearance, we observe that instead in our results the prolonged plantarflexion pushes the model upwards to provide ground clearance for the swinging leg.

However, if we look at the kinematics of the best obtained solution only (instead of averages over multiple runs), we see relatively high correlation of joint angle kinematics for the hip and knee between simulation and human data. We obtain $R_{\text{hip}} = 0.80$, $R_{\text{knee}} = 0.83$. The ankle kinematics show low correlation with human kinematics, $R_{\text{ankle}} = 0.30$. It is clear that although we capture some important aspects of human gait with our optimization, we do not yet explain all aspects using our (relatively simple) model.

Looking at the obtained average and best torques for 10 runs of PSO in Table 4.4, we can see that the *step* and *ppoly* control modes perform slightly better than the *const* control mode, even if there are a significantly larger number of parameters to optimize. Note that the *ppoly* control mode includes the *const* control mode and should therefore in general be able to find solutions performing at least as good. However, due to the increased number of parameters, and the limited number of runs with different initial conditions, we did not obtain better results than the *const* mode.

Figure 4.6 shows the optimized joint reference, stiffness and damping control signals for each of the control modes. Note that the hip and ankle stiffness in the *const* control mode are relatively high, while they are relatively low in the *step* control mode. Furthermore, we see that for both the *const* and *step* control modes, the knee stiffness is very small, being almost only damped (i.e. exploiting the natural leg dynamics). The obtained joint trajectories match the control trajectories for *const* and *step*, but less so for the *ppoly* mode. We can see a clear trend in the stiffness and damping patterns of the *step* mode. Here the hip and ankle joint tend to stiffen during heel-strike, while the damping is optimized for low values during swing (allowing natural swinging motion of the leg). We do not observe these trends as much for the *ppoly* mode.

4.1.4 Discussion

In this work we started by posing three hypotheses about the implications of variable impedance control for human locomotion. We have looked at a minimal model implementation of a humanoid in simulation, and the most simple approximation of energy expenditure. Towards our first hypothesis, we show that human like gaits can be obtained from optimizing for first principles only. Characteristics of nominal human gaits such as heel-strike, foot-roll, toe-off and a 60% stance duration are all observed without optimizing for these characteristics explicitly. Furthermore, best obtained kinematics show a reasonable correlation to nominal human joint angles for the hip and knee joints, although less so for the ankle joint. We have two possible explanations for the mismatch in joint angle kinematics which we should explore in future work. First, our model is only defined in the sigattal plane, and we observe that one of the difficulties is obtaining sufficient ground clearance for the swing leg. It is possible that early during the optimization, solutions are favoured which provide ground clearance in an unnatural way, simply to stabilize the gait. Another possible explanation for the discrepancy is that we use a very simple model of energy expenditure, namely torque. Furthermore, all joints are penalized for torque equally, while it might be more energy costly to generate these torques at distal joints than at proximal joints. A weighting scheme to penalize torques at different joints differently can be explored to see how this affects the obtained gait quality. At the same time, a more sophisticated, biologically inspired model for metabolic cost can be explored.

Our second hypothesis poses that even though we do not explicitly optimize for gait quality, introducing variable impedance implicitly improves gait quality by optimizing only for energy expenditure. Towards this hypothesis we observe some trends, in particular in the *step* control mode in terms of improved joint angle consistency (ankle). Furthermore we observe a relatively low stiffness pattern with a tendency to stiffen the hip and ankle joints during heel strike, and a damping pattern which decreases damping during swing. Furthermore, there is clear evidence for the exploitation of the natural leg swing dynamics by the knee joint, as it shows to have optimized a very low stiffness. During the stance phase, the knee has a significantly increased damping in the *step* control mode which breaks the knee motion. This is also observed during normal human locomotion (Martinez-Villalpando and Herr, 2009) and one of the principles of passive dynamic walking (McGeer, 1990; Wisse, 2004).

Although we expected to see a clear reduction in torque when introducing variable impedance in the controller, we observe smaller improvements in overall torque consumption than we initially expected. We observed a notable reduction of torque in the *step* mode, but did not see the same (or further) reduction for the *ppoly* mode, indicating that our optimizations might easily get stuck in local optima.

This work was published in van den Kieboom and Ijspeert (2013). A video of the resulting optimized walking gait can be found at http://thesis.codyn.net/videos/human_walking.

4.2 CoMan humanoid robot gait optimization

The work presented in the previous section shows that given the (bio)mechanical structure of a human adult-sized person, global human gait characteristics can be obtained from optimization of high-level objectives and simple control laws only. In this section we present work which continues in this direction, looking at several new aspects.

1. We want to know how well this method translates to a humanoid robotics platform. Humanoid robots are certainly inspired by human physiology, but they also have significant differences. In particular, we are interested in looking to apply this method to a model of the CoMan (Compliant Humanoid) robot, which is not only of smaller scale than an adult sized human, but also has different inertial properties, power requirements and importantly, feet.
2. In our previous study we were specifically optimizing for walking at an average – and supposedly energy efficient – walking speed. Doing so would let us simply minimize for overall torque during walking to obtain a sense of energy efficiency. However, for the CoMan we do not have a good estimate of the walking speed at which the system enters an energy efficient mode. We will therefore look at optimizing/discovering the optimal walking speed simultaneously.
3. The objectives previously used to optimize for a human gait included various *artificial* objectives which stabilized the optimization process, avoiding exploitation of unphysical

dynamical behavior. We expected that these issues were caused by the contact model and we address these issues in this section.

4. Finally, we expect that the role of variable impedance control becomes more prominent if we look at perturbations during gait. In the previous section we could see a trend towards variable impedance being optimized around the moment of heel-strike, which can be seen as a particular type of perturbation. Here we will explore this by introducing a periodic force perturbation during the swing phase.

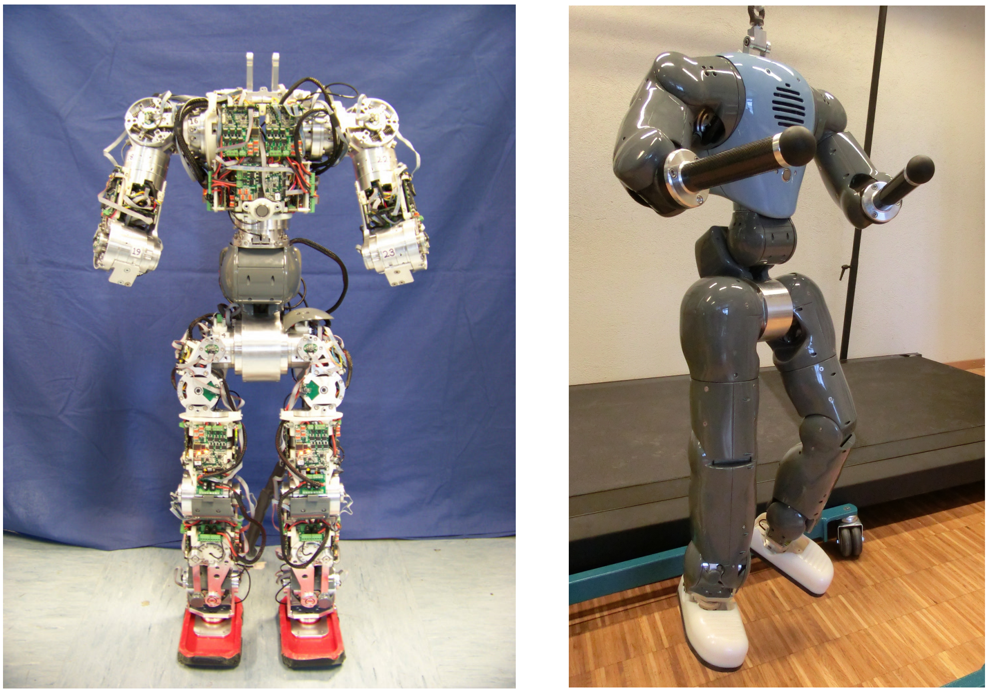
The remainder of this section is organized as follows. First, we will briefly describe the CoMan humanoid robot. Then we will continue by detailing the cȯdγn model based on the robot specifications. What follows then is two studies. The first will look at reproducing the results obtained in the previous section using the same methods, while optimizing walking speed and reducing the objective complexity. Having optimized for walking with the model of the robot, the second study will then look at the influence of perturbations on the optimization of variable impedance control laws.

This work was carried out as part of the WALK-MAN project. WALK-MAN is funded under the European Community's 7th Framework Programme: FP7-ICT 611832. Cognitive Systems and Robotics: FP7-ICT-2013-10

4.2.1 CoMan humanoid robot platform

The CoMan is a child size, humanoid robot which has been developed by the Istituto Italiano di Tecnologia in Genoa as part of the AMARSI European project. Shown in figure 4.7, the robot is approximately 1.2 m tall, comparable to a 6-7 year old human child, and features 23 degrees of freedom, of which there are 6 in each leg. The CoMan, as its name suggests, has been specifically designed to be an intrinsically *compliant* robot using series elastic actuators. The version shown in figure 4.7 has compliant joints in the shoulder pitch/roll, elbow pitch, waist pitch/yaw, and hip/knee/ankle pitch joints. Each actuator has a maximum torque limit of 30 N m and a velocity limit of $6\,\mathrm{rad\,s^{-1}}$. Modeled after a human child, it has the same limb length ratio's as an actual human child. The weight of the robot is approximately 28 kg. Note that the average weight of a human child of the same size is around 22 kg to 25 kg (Cavagna *et al.*, 1983) and the CoMan is thus slightly heavier.

The robot is fully sensorized, with proprioception at every actuator, a 6-DOF force/torque sensor placed under each foot, an IMU at the base of the robot (measuring velocities, accelerations and direction) and finally torque sensors at *each* actuator. This allows the robot to operate in full torque control. This is essential for the implementation of (virtual) impedance control laws on the platform.



(a) IIT version (without covers)

(b) EPFL version

Figure 4.7 – Developed by the Istituto Italiano di Tecnologia, the CoMan is a child size, humanoid robot with series elastic compliant actuators. It has 23 degrees of freedom, among which there are 6 in each leg. a) an earlier version of the robot without its covers (image taken from the IIT website). b) the CoMan version at EPFL.

4.2.2 Modeling the CoMan robot

The model for the CoMan is constructed in the same way as the model of the human in the previous section. The kinematic and inertial data is based on data retrieved from the CAD model of the CoMan, provided by the IIT. Since the data is provided for the full 3D CoMan, we composite all the bodies corresponding to the joints that we do not model in 2D. Figure 4.8 lists the kinematic and inertial quantities of the model. Again, the center of mass locations CoM_x and CoM_y as well as the segment length l are given in proportion to the total robot height (1.2 m). The masses m are given proportional to the total mass of the robot (28 kg).

On comparison with the quantities shown in figure 4.1 (i.e. for the adult size human model), we see that the largest discrepancy between the two models is caused by the center of mass location of the torso. In the CoMan, the center of mass is located much higher than for an average person, which can have a negative impact on the stability. The reason for this is the fact that there are 6 heavy motors located at the very top of the torso (2 of these are deactivated, but present motors to control the neck). Additionally, the torso also contains the control PC and several communication and controller boards.

Another important difference, in particular for locomotion, can be found in the dimensions

Joint	CoM$_x$	CoM$_y$	l	m	r_g
torso	0.00	0.18	$n.a$	0.530	0.11
up leg	0.00	−0.08	0.19	0.132	0.09
low leg	0.00	−0.07	0.17	0.052	0.05
ankle	0.00	−0.02	0.08	0.051	0.03

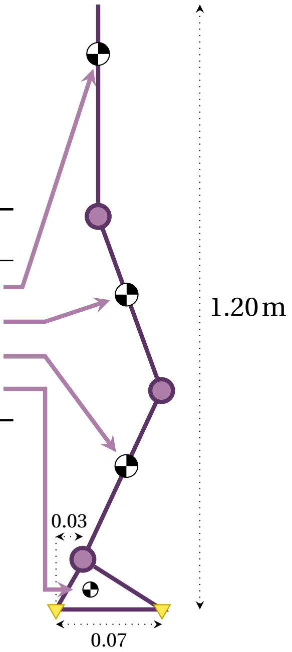


Figure 4.8 – Kinematic and inertial properties of the CoMan model. Kinematic quantities are proportional to the total model height and the mass is given proportional to the total model mass. r_g indicates the segment radius of gyration, from which the inertia can be derived. The schematic on the right shows approximate center of mass locations and contact point locations on the foot (triangles).

of the foot. Comparing l of the ankle (i.e. the distance from the ankle rotation to the sole of the foot) in both figures 4.1 and 4.8, then this distance is around 3% of the total height for adult size humans, but 8% of the total height of the CoMan. The foot on the CoMan is also completely rigid. For this study, which is in the sagittal plane, the rigidity is not important since we use point contacts. however, the length of this rigid foot is 16% of the total robot height. Looking at figure 4.1, we can see that the nominal foot length would be approximately 10% of the total body height. The CoMan feet are therefore significantly disproportionate with respect to its height.

We initially did simulations with a model of the feet corresponding exactly to the dimensions of the feet on the real robot. This however led to unsatisfactory results. None of the simulations resulted in human like gaits, since proper heel-strike, foot-roll and toe-off could not be achieved with such large contact surfaces. Due to the location of force torque sensors in the feet, we cannot easily change the distance from the ankle rotation to the sole of the foot. However, the foot contact plates are easily replaced. We therefore adopted a foot length closer resembling human morphology in our model, as shown in figure 4.8. It should be noted that at the time of writing a corresponding real foot has not yet been manufactured.

Contact modeling

As the hard contact model in cȯdγn was not yet available at the time of the previous study, the soft contact model was used instead (see section 2.6.10). We managed to avoid unphysical behavior of this model by carefully tuning the contact coefficients and adding additional, artificial objectives to penalize unrealistic behavior. This however led to a formulation of the optimization which was more complex than we originally had hoped to achieve.

In this study, we will make use of the hard contact model which since has been available in cȯdγn. As seen in figure 4.8, we also no longer need to model multiple contact points for each foot (which previously made the soft contacts more stable), but instead can model contacts on just the end-points of the two feet. This led to significantly more stable simulations and made an otherwise important tuning step unnecessary.

The basic CoMan cȯdγn model is provided in appendix A (see model A.1).

4.2.3 Study I: CoMan gait optimization

In a first study, we are interested in replicating the results obtained on the adult sized human model for the CoMan humanoid robot. We will address the first three of the objectives listed in section 4.2, namely 1) application of the gait optimization method to a humanoid robot, 2) simultaneous optimization of walking speed and 3) reducing objective complexity. Since there are some significant differences in the physiology of the robot when compared to humans, we expect to see this reflected in the optimized controllers and resulting gaits. By applying the exact same methods as developed earlier, we can compare obtained results directly with our previous study.

Optimization

We will use the exact same optimization procedure as used in the last study, i.e. particle swarm optimization with lexicographic ordering (see section 4.1.2 for a more thorough discussion). Due to the stable behavior of the hard contact model, we can remove the objectives introduced to avoid unphysical behavior from the soft contact model. We have therefore removed the optimization of the stdspeed (walking at a constant speed) as well as the non-contact-time objective (which stated that at least one contact had to be active at all times).

As stated earlier, we do not know a-priori at what speed the CoMan enters a natural mode of locomotion. Instead of optimizing for a specific walking speed, we instead adopt the concept of *cost of transport* (Schmidt-Nielsen, 1972). The *cost of transport* is a non-dimensional quantity which expresses the energy efficiency of transportation, allowing normalized comparison of energy efficiency. It is defined by

$$\mathrm{cot} = \frac{P}{mgv}, \tag{4.2}$$

where P is the power input to the system, m is the total mass, g is gravity and v is the velocity of the system. In our case, we are only comparing the cost of transport between the same systems, and therefore omit mg from our measurement. We thus simply use

$$\mathrm{cot} = \frac{P}{v} \tag{4.3}$$

Note that power is energy W over time, and velocity is distance d over time. An equivalent cost of transport can therefore be obtained from

$$\mathrm{cot} = \frac{W/t}{d/t} = \frac{W}{d} \tag{4.4}$$

For human walking, this energy W is measured from metabolic cost (Anderson and Pandy, 2001). However, since we are looking at a robotics system, and we are not currently interested in electric efficiency of the robot, we use mechanical cost of transport instead. In other words

$$P = \sum_i \tau_i \dot{q}_i, \tag{4.5}$$

where τ_i and $\dot{q}_i$ are respectively the torque and angular velocity of joint i.

Since we allow optimization to find a desired walking speed, we now also need to optimize for the walking frequency. We therefore no longer constrain walking frequency at 0.9 Hz (as described in section 4.1.2). Instead, we allow this frequency to be optimized between 0.4 Hz and 1.2 Hz. The range is intentionally large, to allow for slow (but large) steps and fast (but short) steps.

The lexicographic objectives can then be formulated as shown in table 4.5. Note that we have removed all the artificially introduced objectives. For the second objective, instead of optimizing for a specific speed, we instead optimize for a certain minimum speed. This forces the optimization to be at least moving forward before optimizing the next objective. We emperically observed that without setting a minimum speed, the optimizations would often converge to local optima in which no walking was achieved. We therefore set a minimum speed of $0.3\,\mathrm{m\,s^{-1}}$.

Table 4.5 – Lexicographic Objectives

Objective		Until		
1.	time	time	=	max time
2.	speed	speed	≥	min speed
3.	-assist time	assist time	=	0
4.	-cot	∞		

To enforce plausible solutions with the potential to be applied on the robot, we limit the output torque to $30\,\mathrm{N\,m}$. Furthermore, the maximum controlled stiffness is set to $1500\,\mathrm{N\,m\,rad^{-1}}$ and controlled damping is limited to a maximum of 30 N m s rad^{-1}.

Results

All simulations were run using the same methods as in the previous section. For each control mode (*const*, *step*, *ppoly*), we ran 10 optimizations with random initial conditions.

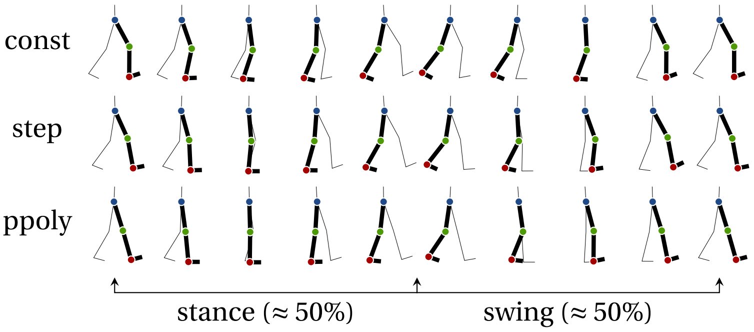


Figure 4.9 – Snapshots of one gait cycle of the best obtained solutions from optimization for each control mode. The gait is shown from heel-strike to heel-strike. Note that gaits are normalized to show a single step, their respective walking speeds are 0.48 m s^{-1}, 0.44 m s^{-1} and 0.33 m s^{-1}.

Figure 4.9 shows snapshots of the best gaits obtained for the three different control modes. There are several important observations to make. First, all three control modes obtain qualitatively human like gaits. Similar to the adult size human optimization, most global human gait characteristics, such as heel-strike, foot-roll, and toe-off are obtained. Both *step* and *ppoly* control modes obtain gaits without knee flexion during stance phase, unlike what can be observed in the *const* mode. Interestingly, all obtained gaits consistently show a stance duration of 50%, whereas during our adult size human simulations we consistently obtained a stance duration of 60%. This difference can be explained by the lack of a toe, which in the human model would prolong the end of the stance phase.

With regard to walking speed, the *const* and *step* control modes both optimize for relatively faster walking (0.48 m s^{-1} and 0.44 m s^{-1} respectively), as compared to the *ppoly* mode (0.33 m s^{-1}). Table 4.6 shows obtained frequency, speed, step length and cost of transport for the best solutions of each control mode. The best gait, in terms of cost of transport, is obtained using the *step* controller, while the *ppoly* controller shows significantly worse performance. The obtained walking speeds and corresponding frequencies are very similar between the *const* and *step* controllers, and the *natural* walking speed of the CoMan (from a mechanical point of view) seems to be around 0.45 m s^{-1}, or 1.6 km h^{-1}. See B.1 in appendix B for a rendering of the walking sequence of the optimized *step* controller.

Compared to data measured from children (Cavagna *et al.*, 1983), the obtained gaits here are much slower and produce larger steps than observed normally in children of the same size as the robot. We attribute this difference mainly to the difference in center of mass location of

Table 4.6 – Gait speed characteristics of best obtained solutions

Type	**Frequency** (Hz)	**Speed** ($\mathrm{m\,s^{-1}}$)	**Step length** (m)	**Cost of transport**
const	0.64	0.48	0.74	40.5
step	0.63	0.44	0.69	27.8
ppoly	0.53	0.33	0.63	64.9

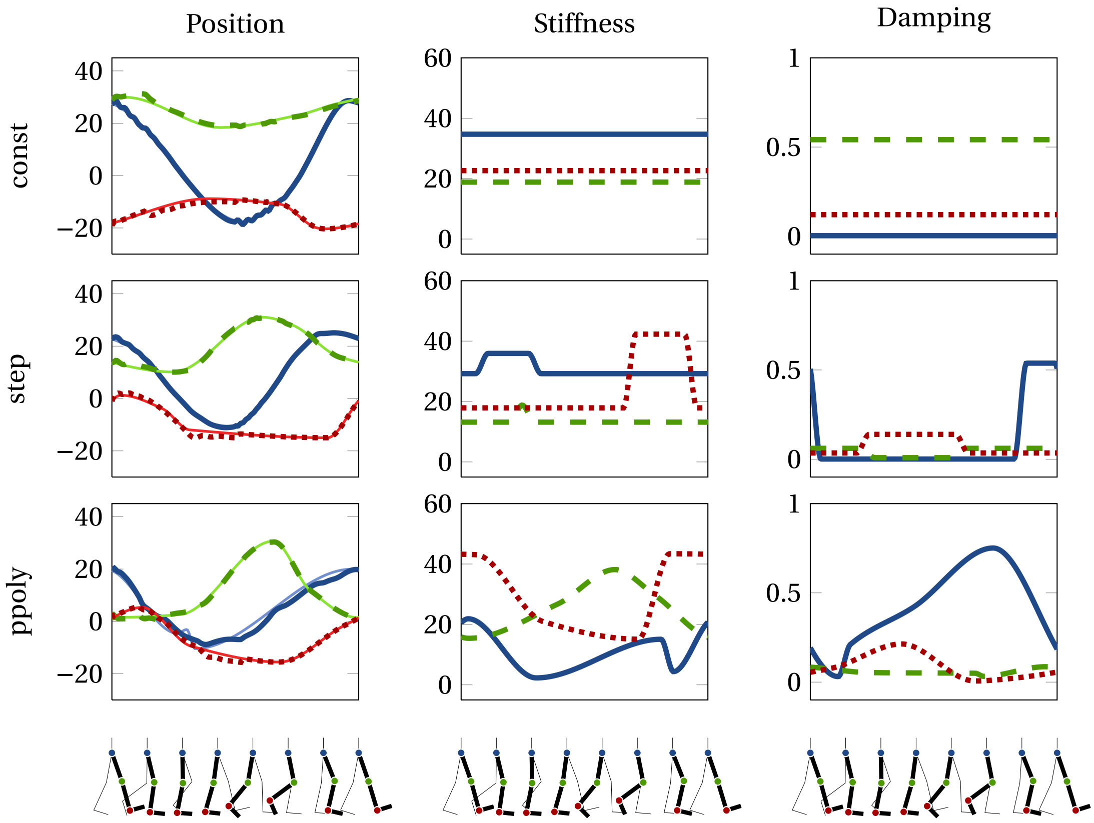


Figure 4.10 – Optimized joint control signals. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$.

the torso (which is much higher than it should normally be) and the torque limitations of the platform.

Figure 4.10 shows the control signals, reference position, stiffness and damping optimized for the three control modes. Note that the desired reference joint angles are closely followed by the actual joint angles. The resulting control therefore seems to be largely kinematic. We do not observe exactly the same trends as in the previous study with respect to the obtained stiffness patterns. Previously, we noticed a trend towards increased stiffness at heel-strike, presumably to stabilize the ground reaction forces during impact. However, due to the hard contact model, we do no longer observe this trend.

Figure 4.11 shows the control output torques corresponding to the impedance control law output from the control signals. As can be seen, most output torques stay within the CoMan actuator limits. In particular, the output torques for the *step* controller look promising when taking torque limits into account. A substantial amount of torque is required on the ankle during mid to end stance. This is to be expected, since the ankle has to push the body forward. However, these torques could possibly be reduced by a better designed foot, in particular the distance from the ankle joint to the ground.

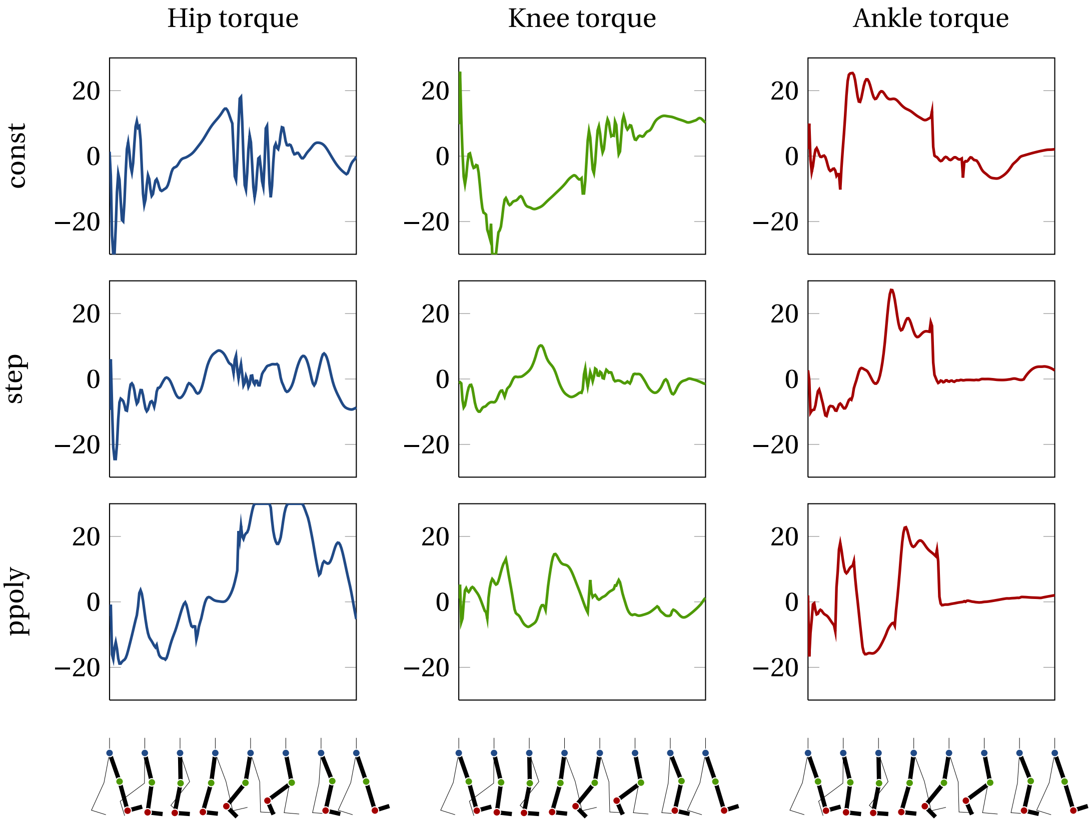


Figure 4.11 – Resulting control torques for the best solutions obtained for the three control modes. Torques are within limits in most cases, except for the hip torque in the *ppoly* mode and the knee in the *const* mode. The *step* torques are within range of the limits for all joints. Large torques are consistently required during mid to end stance for the ankle.

Discussion

The objective of this first study was to use the methodology developed in section 4.1 to optimize a natural gait for a humanoid robotic platform. We successfully did so while 1) reducing the complexity of the objective function such that only high level objectives remain, and 2) no longer optimizing for a specific, known walking speed. This shows that the method is both robust and does not depend on strictly close-to-human properties of the system. Since we have shown in our previous study that the method is a reasonable model for human walking, we can conclude with some certainty that the obtained results for the CoMan platform are equally reasonable.

We have also shown that although qualitatively well performing gaits can be obtained for the CoMan, it remains to be seen if the obtained controllers can be transferred to the robot. It is clear that the design of the platform has not been guided by locomotion performance, in particular when looking at the torso mass distribution and the current design of the foot. Although it would be difficult to lower the torso mass, it would be worth investigating the design of new feet specifically suitable for locomotion. We further believe that the presented methodology of optimization of human like gaits could aid in this effort by validating or even co-designing (as we will see in chapter 5) new feet in simulation.

A video of the resulting optimized walking gait for the CoMan robot can be found at http://thesis.codyn.net/videos/coman_walking.

4.2.4 Study II: The effect of impedance control during perturbations

In the previous section we have seen that the developed methodology for optimizing for humanoid gaits using impedance controllers by use of particle swarm optimization with lexicographic ordering can be used effectively to optimize for human like gaits for the CoMan humanoid robot platform. Although we have hypothesized that impedance control can contribute to the stability and robustness of the gaits, we have not been able to find significant evidence for it in our previous studies. In hindsight, this might not be surprising. Even though we observe certain trends towards improved performance (in particular when using the *step* controller), variable impedance cannot be exploited during steady state.

Our next hypothesis, then, is that variable impedance becomes more prominently useful in the face of *non*-steady state locomotion. When looking at impulsive perturbations (i.e. non prolonged perturbations), such as force pushes, intuitively speaking it is often better to be compliant in the direction of the perturbation. Doing so allows for minimization of the dynamical impact of the perturbation. Or so we expect. To test for this hypothesis, we devise a new set of experiments which optimize variable impedance controllers for locomotion under periodic perturbation.

Perturbations

The type of perturbations under which we will study the effect of variable impedance control is a periodic, impulsive force perturbation on the swing leg. In our case, it is important that the perturbation is periodic since we are still optimizing time dependent signals. We expect to see a clear effect of impedance modulation during the period in which we apply the perturbation forces. Furthermore, a force perturbation during the swing phase could intuitively benefit from a variable impedance controller. We hypothesize that we will observe reduced stiffness during the period of possible perturbation, such as to give in to the impulsive force without disturbing the stance dynamics.

Force perturbations are applied on the foot in the horizontal direction. The magnitude of the force and the duration of the impulse are sampled from a normal distribution. In this study we use a force magnitude of $\mathbb{N}(80,5)$N and a force duration of $\mathbb{N}(0.15,0.05)$s (where $\mathbb{N}$ indicates a normal distribution). The onset of the perturbation is sampled from a uniform distribution each time a leg goes into swing phase.

Optimization

We use the same optimization methodology as used in the previous section. However, since we are now subject to stochastic perturbation forces during locomotion, we need a procedure by which to ensure that the performance of a particular controller is not dependent on the specific stochastic behavior during a single run. Therefore, we run each controller N times using a different random seed and thus obtain different perturbation forces. We then use the *worst* performance as the final objective value of a solution. In all experiments, N was set to 3.

Since we have previously obtained various controllers which perform state-stead locomotion, we will here reuse this information to bootstrap the optimization under perturbation. To do so, we take the 3 best solutions from each previous simulation run, resulting in a total of 30 solutions with known initial conditions in the initial population of the perturbation study. The remainder of the population is created from this initial population by applying random mutations on all parameters in a uniform manner. We have used a mutation magnitude of 10% for all parameters for this study. The resulting population should ideally be in the vicinity of potential solutions, or at least more so than when starting with a fully randomized initial population.

Results

We begin by looking at the fitness progression of the runs with the best results for each of the three control modes. In particular, we look at the amount of time spend and progression of the last lexicographic objective (i.e. optimization of cost of transport) for each control mode, which is shown in figure 4.12.

There are a few important observations which can already be made from looking at these graphs. First, the *step* control mode spends far more time optimizing for the cost of transport, i.e. the final objective, than the other control modes. In fact, *ppoly* hardly manages to reach the final objective, failing to stabilize the gait. Table 4.7 summarizes these observations.

What is interesting to observe here is that the best obtained cost of transport follows the same relative trend as observed in our previous study, however is approximately 5 times larger as before. Since the locomotion speed has only changed marginally, this means that there is a

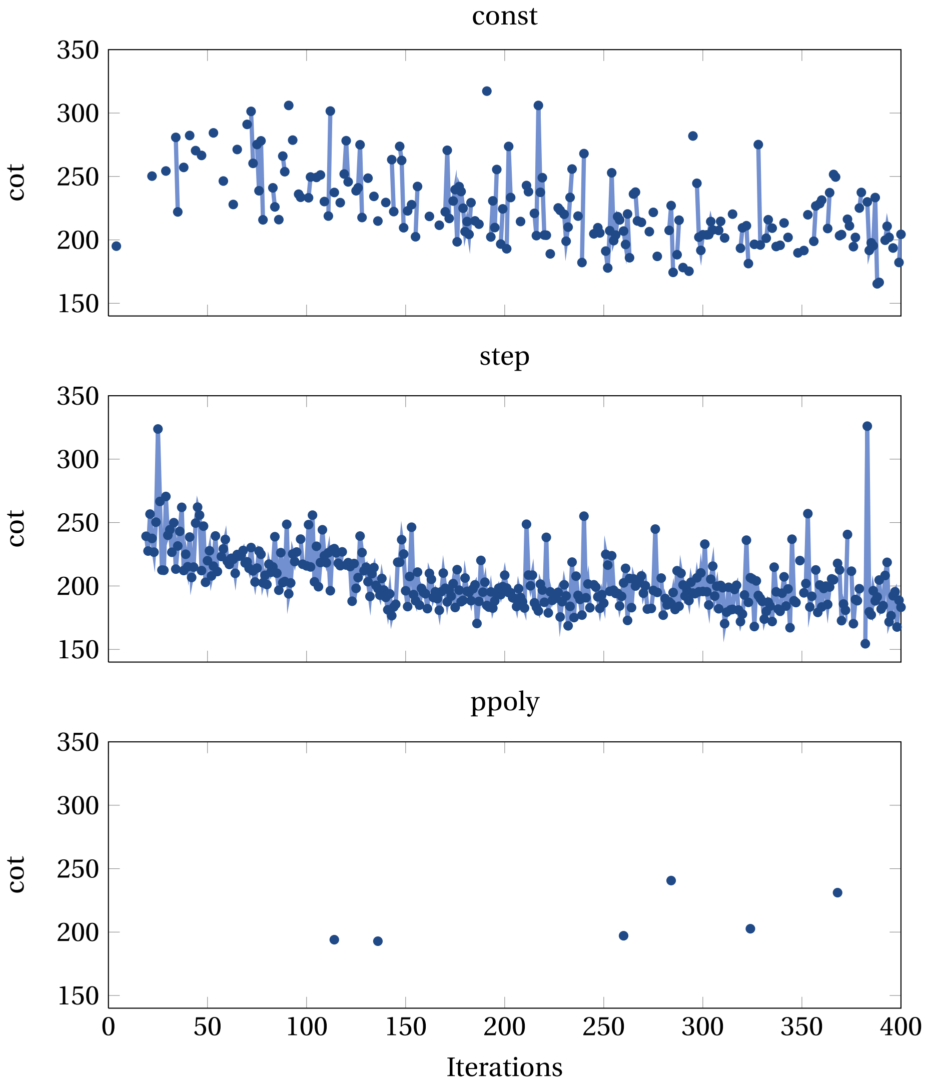


Figure 4.12 – Fitness progression while optimizing for the last lexicographic objective, cost of transport, for the run with the best obtained fitness for each control mode. From top to bottom, the progression of respectively *const*, *step* and *ppoly* is shown.

Table 4.7 – Optimization summary

Type	**Best cot**	**Best speed** ($\mathrm{m\,s^{-1}}$)	**Average % final objective**
const	165	0.37	14%
step	154	0.38	58%
ppoly	192	0.3	9%

significant increase in power consumed due to the applied perturbations.

When looking at the reason why the *const* and *ppoly* optimizations spend a significantly smaller amount of time optimizing for cost of transport we see that those control modes are not able to consistently handle the perturbations. Even though each solution is evaluated 3 times, keeping its worst performance as the final fitness, when we rerun those solutions with different perturbations we do not obtain again the same performance. In fact, most of the time, those solutions fall back to the *assist time* objective, unable to self-stabilize. This is not the case for the *step* control mode which shows much more reliable performance under perturbation. Due to the stochasticity of the perturbations, solutions which were performing well before, might not do so in the future. However, standard Particle Swarm Optimization does not incorporate this into the optimization and thus might be attracted to solutions which are in fact performing less than previously observed.

Although this is a problem which could be addressed, it is not the primary objective of this study. The problem occurs for all control modes, and we thus are not biased by it towards a particular mode. The main objective of this study is to determine whether the advantage of joint level variable impedance control emerges while under perturbation without explicit optimization for it. We therefore turn to look at the optimized control patterns for the *const* and *step* control mode.

Figure 4.12 shows the control signals for the best solutions of the *const* (left) and *step* (right) control modes. The most interesting observation here can be made when looking at the stiffness and damping patterns of the *step* mode. Unlike in our previous studies, here the stiffness and damping patterns show a marked correspondence with the perturbation signal. The change between the two values of stiffness and damping shows a clear correlation with the window of possible perturbations.

In particular, we see that the stiffness of the knee *increases* (slightly) during the period of possible perturbation, while the stiffness of the hip shows a large *decrease* at the end of perturbation. The opposite is observed for the damping , which *increases* for the knee during late swing, while it *decreases* for the hip during early swing.

The obtained correspondence between the perturbation and the change of impedance is clear, and this particular solution has optimized in particular a decrease in hip stiffness and damping during period of the swing phase where perturbations can occur. Figure B.2 in appendix B shows a single step from the walking sequence of the *step* controller as shown in figure 4.13.

A video of the walking gait under perturbation can be found at http://thesis.codyn.net/videos/coman_perturbation.

4.3 Conclusion

In this chapter we have explored the optimization of natural human gait from high-level objectives, such as walking at a certain speed while minimizing for a measure of energy expenditure. We showed that using Particle Swarm Optimization (see section 3.1.3) and lexicographic or-

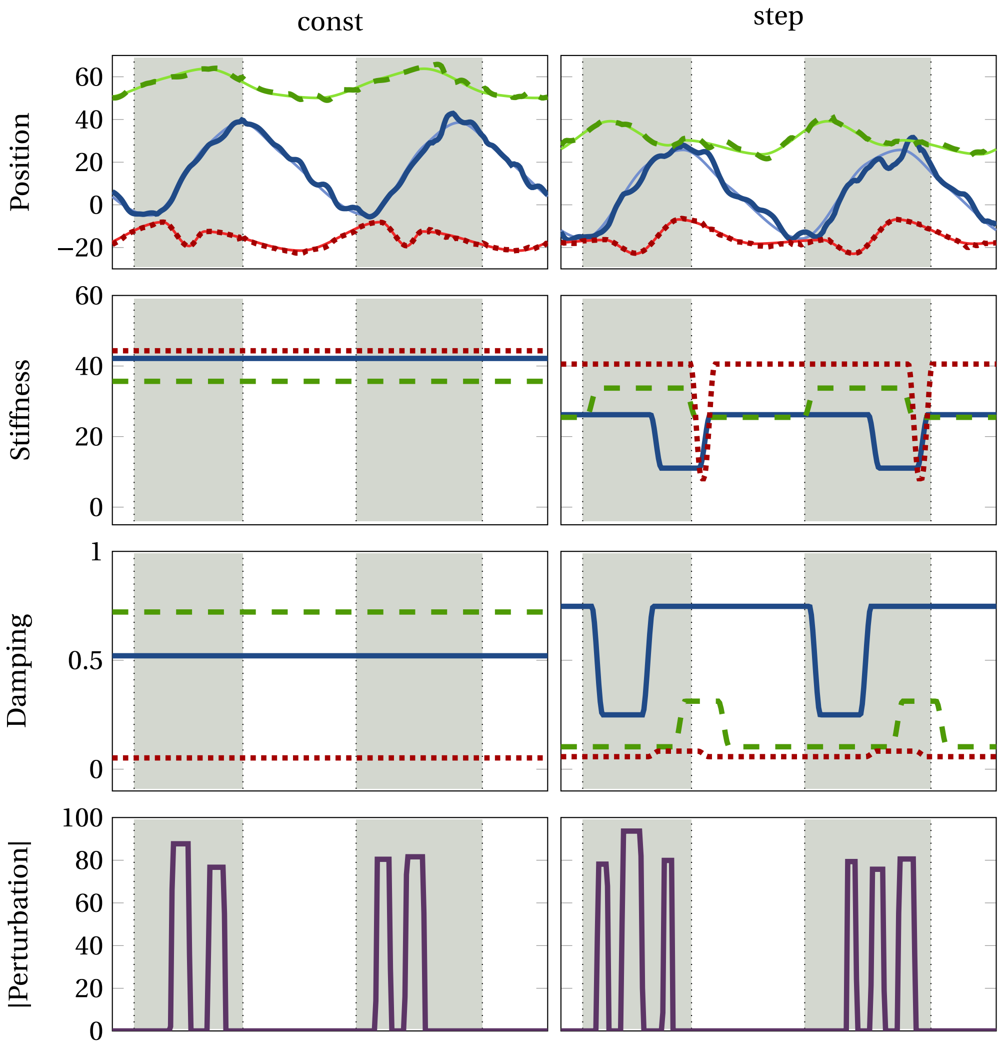


Figure 4.13 – Optimized joint control signals of the best solutions for the *const* and *step* control modes during the last two periods of locomotion. The hip, knee and ankle signals are shown in blue (solid), green (dashed) and red (dotted) respectively. The position plots show both the reference signal (thin solid) and the resulting (measured) joint angle. Joint angles are shown in degrees, stiffness in $\mathrm{N\,m\,rad^{-1}\,kg^{-1}}$ and damping in $\mathrm{N\,m\,s\,rad^{-1}\,kg^{-1}}$. The magnitude of the perturbation force during walking, as applied to the ankle, is shown in the bottom plot. The shaded areas indicate the swing phase.

dering (see section 3.3.2) of optimization objectives, a natural, stable gait corresponding to an adult sized human model can be obtained. We hypothesized that variable impedance control could improve performance of the obtained gaits, but only found small trends towards improved performance using a simple step variable stiffness and damping controller.

Having verified that our proposed optimization method works well to (re)discover natural human gait for an adult sized human model, we applied the exact same optimization methodology (albeit with different objectives) to a humanoid robotic platform, the CoMan in section

4.2. Although this platform resembles a child size human, several of its morphological properties differ from that of a human person. In particular, the feet and the center of mass of the robot are significantly different. We show that by changing the dimensions of the feet, we can again obtain natural gait while additionally optimizing for the cost of transport instead of torque requirements. This allows the optimization to find the optimal speed to power ratio for this particular model.

Finally, in our last study in section 4.2.4 we investigate further the possible role of variable impedance control by introducing perturbations on the swing leg during locomotion. By making the perturbations periodic, we can still optimize for an open-loop variable impedance controller allowing for direct comparison with our previous studies. We show that indeed under perturbation, the role of variable impedance becomes significant, leading to more reliable perturbation rejection by reducing stiffness and damping of the hip during the expected period of possible perturbation. It should be noted that we do not assume to reliably obtain conclusive results on the type of impedance modulation necessary for perturbation rejection. Rather we observe and conclude that allowing optimization of modulation of impedance allows for improved robustness against rejection of perturbation.

Although using an open-loop controller allowed us to systematically explore the role of variable impedance, it is clear that an actual controller should modulate impedance based on feedback of an observed perturbation. There is little gain to be had from variable impedance during steady state locomotion, but our last experiments showed that modulation of stiffness and damping could provide for a simple mechanism to reject (although marginally) certain perturbations and that our optimization method is able to exploit the impedance modulation properly. Future work would include studying feedback controllers to modulate impedance based on sensor information, which could further increase locomotion robustness.

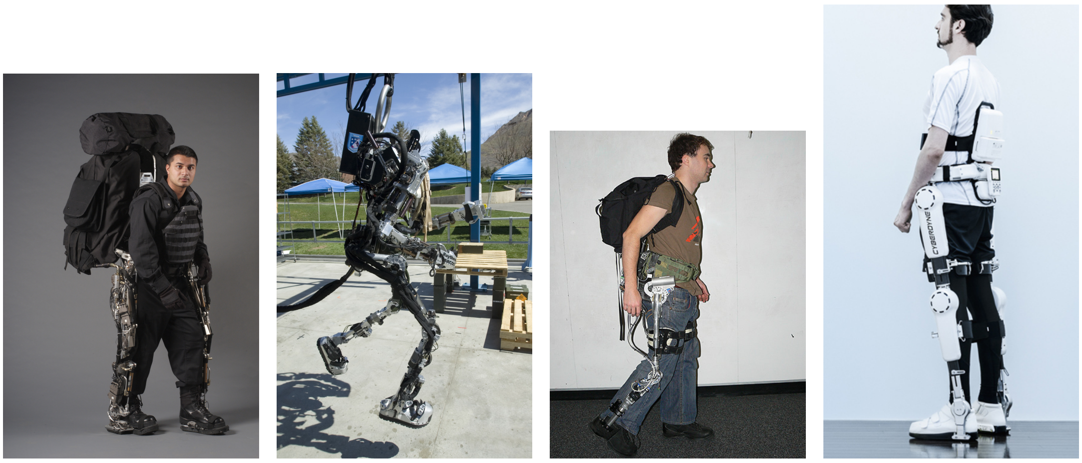
Finally, we only show results here obtained in simulation. It remains to be seen if these same results can be obtained on the real platform. There are of course additional difficulties for transferring our controller. First, all our simulations are done in 2D, while the robot requires a controller for the third dimension. Second, we obtain marginally stable gaits in simulation since we do not explicitly control the global stability of the robot. Even though the perturbation study shows that the controller is able to self-stabilize under certain perturbations, we do not expect this to be sufficient when applying the same controller on the real robot. One possible direction would be to use our controller as a nominal pattern generator while providing an additional stabilization controller which modulates this pattern such that the complete control becomes globally stable.

5 Co-design of human assistive devices

Having developed all of the necessary methods for the modeling of dynamics, optimization of continuous parameters within discrete sets of solution classes, and having investigated the role of impedance control for (re)discovering nominal human gait, we now turn to the last topic of this thesis. Although there are many different areas to which robotics research can contribute, few are of arguably greater potential impact on society than those concerned with the development of wearable robotics such as prosthetics and orthotics research. This is not necessarily a new field of robotics research, since many successful prostheses and orthoses exist today, both for the lower and upper extremities. Most of these are however either purely passive, or have only limited powered capabilities. There has been a recent increase in the research and development of active or powered wearable devices, now that technology has enabled more compact actuators and higher power density energy supplies.

We are particularly interested in the development of powered, wearable robotics for the assistance of the lower extremities. Lower extremity wearable robots can be divided into two main categories, those meant for 1) the augmentation of an able-bodied person and 2) for the assistance of a person suffering from an impairment of the lower limbs (for example hemi- or paraplegics). Several wearable robots have been developed in both categories in recent years.

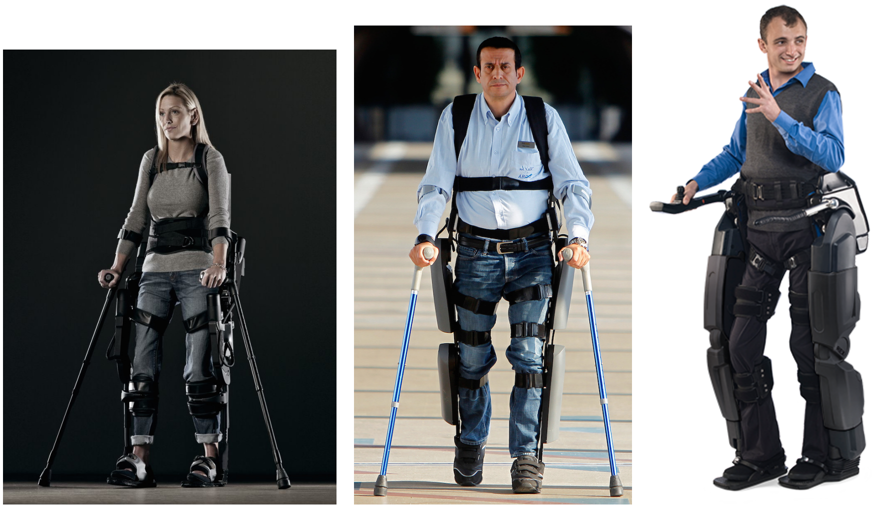
In the first category, probably the most well known examples include the BLEEX (Berkeley Exoskeleton) (Kazerooni and Steger, 2006) (figure 5.1a) and the Sarcos (figure 5.1b), both of which were a result of a DARPA sponsored program, Exoskeletons for Human Performance Augmentation. Both of these exoskeletons are capable of above human performances in terms of load bearing and weight carrying. Although one of the primary motives for the development of these type of exoskeletons is in the military, they could also be used for industrial purposes, or disaster scenarios. A different approach was taken by the MIT exoskeleton (Walsh *et al.*, 2006, 2007) (figure 5.1c), which focused on a semi-passive design in which passive dynamics were exploited in an effort to create a much easier to wear and more energy efficient exoskeleton. Finally, the HAL-5 exoskeleton (Guizzo and Goldstein, 2005) (figure 5.1d) is another exoskeleton which takes a different direction yet, in which control is derived from measurement of EMG of the operator. Although initially shown in demonstrations for load carrying, it has been



(a) BLEEX (b) Sarcos (c) MIT (d) HAL

Figure 5.1 – Existing exoskeletons for human augmentation. From left to right a) the BLEEX exoskeleton (source http://bleex.me.berkeley.edu/), b) the Sarcos exoskeleton (source http://robohub.org), c) the MIT exoskeleton (source Walsh *et al.* (2007)) and d) the HAL exoskeleton (source http://www.cyberdyne.jp/)

designed specifically for the assistance of impaired and elderly persons during daily tasks. It is also one of the first exoskeletons to be successfully commercialized and has recently been used in a number of hospital settings.



(a) Ekso (b) ReWalk (c) Rex

Figure 5.2 – Lower extremities orthosis used for rehabilitation and support of paraplegics. From left to right a) the Ekso by Ekso Bionics (source http://www.eksobionics.com), b) the ReWalk from Argo Medical Technologies (source http://rewalk.us, courtesy of Argo Medical Technologies) and c) the Rex from Rex Bionics (source http://www.rexbionics.com)

This brings us to the second class of wearable robots, those used for rehabilitation or assistive purposes. The aim of these is not to elevate human performance, but rather provide assistance or support to those who are impaired or in need of rehabilitation. This is arguably of greater sociological importance than the creation of “superhumans”, and there are approximately 5 million people in the United States alone who could benefit from a lower extremities orthosis. There are a few recent success stories in the development of exoskeletons for the assistance of paraplegics allowing standing and even walking. The Ekso (formerly called eLegs) is being developed by Ekso Bionics (previously Berkeley Bionics), and provides paraplegics with the ability to walk again (figure 5.2a). Crutches have to be used for stabilization since the device does not provide self-stabilization at the moment. The ReWalk (figure 5.2b), from Argo Medical Technologies, employs very much the same design as the Ekso, as both use electrical actuators (instead of hydraulic, which most human performance augmenting exoskeletons use). Similar to the Ekso, paraplegics can use the ReWalk to walk with crutches for stabilization. Wearable robots which self-stabilize, i.e. do away with the need for crutches, are also being developed. The Rex (figure 5.2c), developed by Rex Bionics, is a slightly bulkier wearable robot, but provides full independent movements for users who were previously restricted to using a wheelchair.

All of the above mentioned exoskeletons, whether used for carrying large loads, rehabilitation or assistance all have one thing in common. Their design is anthropomorphic, i.e. they follow human morphology exactly. There are many advantages to such a design. First, it is a relatively simple design from a kinematic perspective since segment lengths and joint locations are known beforehand. Secondly, the acceptance of an exoskeleton which follows the human limbs is arguably much higher than for a possibly non-anthropomorphic one, i.e. people feel socially more comfortable wearing a device that does not significantly alter their appearance. On the other hand, some devices, such as the Flex Foot Cheetah (Nolan, 2008) have received much attention and although not anthropomorphic, are generally viewed positively. However, here we focus on social acceptance in the nominal case, where generally people prefer to be “normal”. Of course, what is socially acceptable is a contemporary notion and changes over time.

On the other hand, a non-anthropomorphic design for a wearable robot could have significant advantages on its own. Such devices have more kinematic freedom and have the potential of providing improved dynamic behavior with regard to interaction with the user. This additional level of design freedom can improve user comfort (Schiele and van der Helm, 2006) by avoiding issues such as joint misalignments causing uncomfortable interaction forces between the wearable device and the person. Of course, they are also inherently more complex in their design, requiring more joints, connecting segments and overall mechanics. Additionally, special care has to be taken to avoid singularities which are bound to occur in non-anthropomorphic parallel structures.

Since the design of such devices can be difficult and non-intuitive, we aim in this work to provide a methodology for the iterative design of non-anthropomorphic, lower extremities wearable robots through the use of evolutionary inspired optimization algorithms, co-designing both the wearable robot structure as well as its control. There are many important aspects to the design of a wearable device, aimed to be used for rehabilitation or assistive purposes. Although we are fully aware of the importance of device interaction interfaces with the user, device aesthetics for general acceptance and user adaptive control, we limit ourselves in this study to the development of a methodology for the design of the mechanical structure and non-interactive control only. See Ronsse *et al.* (2010, 2011a,b,c) where we explore adaptive control for wearable robot assistance using central pattern generators.

In the following sections we first describe our initial studies towards developing our methodology. Then we continue to detail the experimental design method, modeling of the non-anthropomorphic wearable robot, specifics of the optimization procedure and finally the obtained results from this study.

The work presented here has been carried out as part of the EVRYON European project from the European Community's Seventh Framework Programme FP7/2007-2013 - Future Emerging Technologies, Embodied Intelligence, under the grant agreement no. 231451 (EVRYON).

5.1 Initial study

In the seminal work of Karl Sims (Sims, 1994b,a), creatures were evolved to perform certain tasks in a competitive environment. The novelty was that not only were the creatures' controllers evolved, but their morphologies were as well. It showed that on a variety of different tasks, such as walking, swimming and jumping, strategies would emerge, both in morphological terms and control, which would be hard to design manually. Similarly, in Hornby *et al.* (2001) stick like robotic lifeforms were evolved using Lindenmayer systems (Lindenmayer, 1968) in which repetitive structures can be encoded with a small number of parameters.

It has been shown that co-design of morphology and control can be beneficial in terms of optimizing for the efficiency of a system. In Paul and Bongard (2001) it was shown that when optimizing for the morphology of a simple bipedal walker, increases in performance could be observed with respect to the original bipedal morphology. Here we hope to apply these same principles to the design of a wearable robot.

When we first started to explore the feasibility of this approach we simply tried optimizing for the morphology and control of bipedal like "creatures", only concerned about forward locomotion. The results can be seen in the movie provided at http://thesis.codyn.net/videos/codesign/crazybipeds.mov. We then did the same thing, only this time for evolving quadrupedal creatures, for which the results can be found at http://thesis.codyn.net/videos/quadruped.avi. Of course, these were simple "toy" like example optimizations, but there are some important observations to be made:

1. As the first movie shows, allowing the optimization of morphology can lead to surprising and unintuitive solutions for the task of locomoting forward.

2. Many different solutions exist which, while performing the same in terms of locomotion speed, achieve this in very different ways.
3. If the only goal is to locomote forward, many solutions emerge which could never be realized due to joint limits, fast motion, inter collisions, etc. Therefore, care has to be taken to optimize for the appropriate objectives to obtain a desired result, and with the right constraints.
4. It is possible to retrieve, without prior specification of a specific gait, a natural locomotion gait as in the case of the quadruped in the second movie. Here we did not specify beforehand the phase between the limbs, however a natural gallop appears in which the morphology has evolved such as to form a primitive foot support structure.

This initial study verified that it is possible to obtain surprising and novel methods of locomotion while at the same time not preventing the occurrence of natural locomotion, if it turns out to be the most efficient solution (according to the objectives optimized for).

After this initial experiment, we continued with developing the co-design of a wearable robot structure for human assistance. In that study (of which the details are not presented here) we used Webots (Michel, 2004) (an ODE based simulator) to simulate human models with augmented parallel structures for the task of locomotion. Analyzing the results from these simulations, we observed that although structures were able to locomote properly, various issues prevented the use of these structures as the basis for a prototype of a wearable robot. In particular, optimizations almost always made use of mechanically singular configurations in order to obtain efficient locomotion, which was highly undesirable for the real device. Additionally, we also observed that optimizations could lead to the occurrence of large internal forces on the human model joints. Unfortunately, ODE does not allow for reliable determination of these internal forces. This made it difficult to incorporate minimization of these forces as an objective during optimization, since a physically representative threshold could not be set correctly.

The results of these simulations can be found at http://thesis.codyn.net/videos/codesign/first. These models immediately show the difficulties of using these type of optimizations for the co-design of a wearable robot. Most of the solutions would either be hard to realize realistically, would involve fast moving parallel (sub)structures or would not be acceptable to wear in a social environment. This also shows that, as suggested in Paul and Bongard (2001), this type of design methodology is better used as an iterative design input, rather than a final solution. Of these early simulations http://thesis.codyn.net/videos/codesign/first/exo823_fast.avi is of interest since it features a structure which is close to the body but nevertheless is of an intricate parallel design which leads to reasonably efficient transfer of energy from the wearable robot joints to the human body. Nevertheless, it also made use of singularities in the kinematic structure which is why it was finally discarded along with the other solutions.

These results set us on a new path. Intrigued by the possibilities of a non-anthropomorphic design, we set out to develop a robust, reliable and open simulation environment in which we could iteratively design these type of structures without the restrictions imposed by existing simulation software. This finally lead to the development of cȯdγn which provides all of the necessary tools for modeling and simulating closed loop parallel structures, measuring of constraint forces and accurate contact models. In the remainder of this chapter we will describe the co-design of a non-anthropomorphic wearable robot for the lower extremities using our previously developed tools.

5.2 Design

The co-design of both the morphology and the control of a wearable, lower limb robot for the task of steady state locomotion is a complex optimization problem, especially when taking an open-ended approach as proposed here. The approach that we adopt is to develop a methodology for the co-design of wearable devices providing an open-ended and iterative method to automatically explore possible designs and complex structures. However, the result of this co-design is not a finalized product, but rather provides insights into the possibilities of new designs that would have otherwise been difficult to achieve. To keep the problem tractable, we simplify the process by imposing a number of constraints and some assumptions. We believe that these assumptions do not diminish the validity of the approach, although it does indicate that there are certain limitations to the scale at which our method can be applied. We will discuss this in more depth in section 5.6. More specifically, we introduce the following constraints and assumptions:

1. *Sagittal*: As with our previous studies, we will only consider the sagittal plane, i.e. we focus on walking in the 2D plane. Additionally, we ignore for the moment inter collisions between segments of the human body and the wearable robot under the assumption that there are ways to offset parallel segments in the y-direction.
2. *Joints*: We only consider a wearable robot for actuating the *hip* and knee, leaving the *ankle* unactuated by the wearable robot. Although the *ankle* is important during walking, we assume that it could be actuated anthropomorphically, and the design is not part of this study.
3. *Structure*: Wearable robot joints are connected through simple, rigid bars and the number of degrees of freedom, per leg, is exactly 2. We consider the construction of the wearable robot from 3 joints and 4 segments. Note that more complex structures (i.e. with more joints and more segments) could also be considered without loss of generality. Here we follow constraints imposed by EVRYON.
4. *Actuators*: We assume perfect torque actuators and ignore possible effects from the motor dynamics. Since there are two DOFs, we will have two actuators per leg.
5. *Attachment*: The wearable robot is attached to the human body in a perfectly rigid manner. The interaction interface between the human body and a device is important, both ergonomically and because it is responsible for transferring forces for a real device. Here however we choose not to focus on this aspect of the design.

As can be seen from this list, there are a fair number of important aspects of wearable robot design that we do not choose to address. We have a strong focus on the exploration of robot morphologies and the design methodology, while focusing less on the practical implications of building such a system.

To continue, we will first look at the family of wearable robots that are the result of our design choices listed above, in the form of their topology and morphology.

5.2.1 Topologies and morphologies

Since we are looking at the design of a non-anthropomorphic wearable robot, we start by defining attachment points on the human body to which the parallel structure of the robot can be attached. Since we are aiming to actuate the human hip and knee joints, we define three attachment locations as shown in figure 5.3: One on the torso, one on the upper leg, and one on the lower leg.

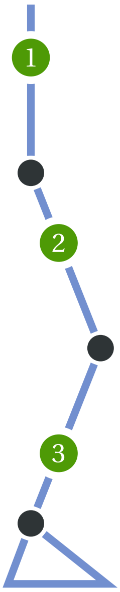


Figure 5.3 – Schematic representation of the attachment joints at which the wearable robot can be attached to the human body.

We can look at the definition of the structure of the wearable robot in terms of its *topology* and its *morphology*. The *topology* of a structure determines the number of degrees of freedom and the manner in which they are connected. It therefore uniquely identifies the *connectivity* of all the joints and segments in the system and can be represented by a graph. The concrete realization of a certain topology into a structure results in a *morphology*. This realization involves choosing the geometrical properties of the joints and segments, such as the length of each segment and the placement of actuated joints. Figure 5.4 illustrates this principle schematically.

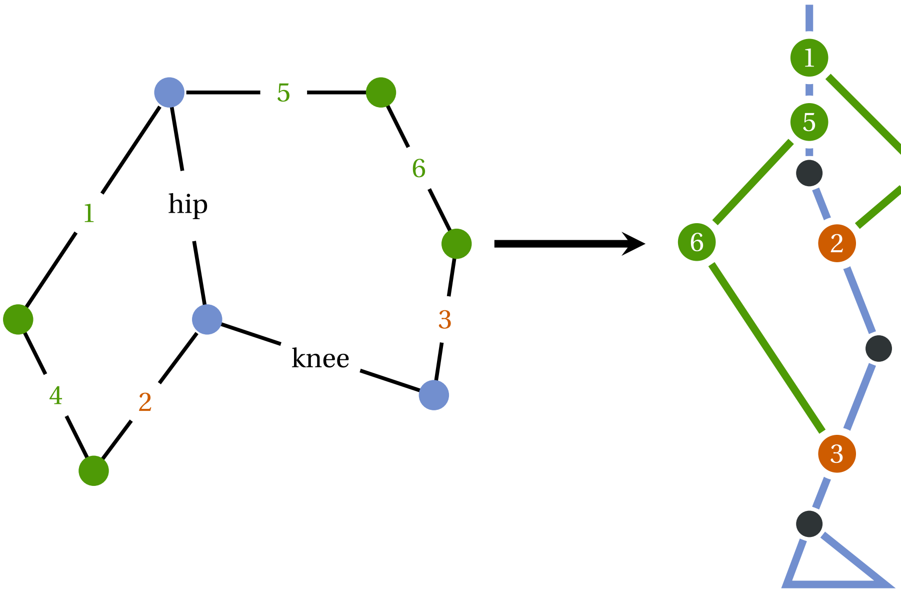


Figure 5.4 – Schematic representation the realization of a specific *morphology* (right) from a kinematic *topology* (left). On the right, the *topology* is represented as a graph where edges represent joints and vertices represent segments. One possible realization of the *topology* is shown on the right. Joints in orange are the loop closure joints.

In Sergi *et al.* (2011), a method is developed for the exhaustive enumeration of wearable robot topologies for a given number of wearable robot segments, joints and a desired number of degrees of freedom. Topologies can be represented by an adjacency matrix which encodes the connectivity of the segments in the structure. Considering the fixed kinematic structure of the human (i.e. the connectivity of torso, hip, knee and ankle), the wearable robot attachments and the number of additional wearable robot segments and joints, all possible adjacency matrices can be enumerated, resulting in a desired mobility. The mobility of a system with parallel structures is defined as the resultant number of DOFs in the system, and is defined for planar structures by (Kutzbach, 1929):

$$k = 3(l-1) - 2n, \tag{5.1}$$

where l is the number of segments and n is the number of joints in the structure.

In our case, it is useful to separate the mobility into human, attachment and wearable robot:

$$\begin{align} k &= 3(l-1) - 2n \tag{5.2} \\ &= 3(l_h + l_{\mathrm{wr}} - 1) - 2(n_h + n_a + n_{\mathrm{wr}}), \tag{5.3} \end{align}$$

where l_h is the number of human segments (i.e. 3), l_{wr} is the number of wearable robot segments, n_h is the number of human joints (i.e. 2), n_a the number of attachment joints (i.e. 3) and finally n_{wr} the number of wearable robot joints. Considering a mobility of $k = 2$, we can then express this equation in terms of l_{wr} and n_{wr}:

$$6 = 3l_{\mathrm{wr}} - 2n_{\mathrm{wr}} \tag{5.4}$$

Given this equation, the minimally interesting set of topologies where $k = 2$ is given when $l_{\mathrm{wr}} = 4$ and $n_{\mathrm{wr}} = 3$, which is the set that we will consider here. The method as described in Sergi *et al.* (2011) first generates all of the possible combinations of connectivity adjacency matrices. Then, it prunes this set by removing solutions which are over-constrained, contain disconnected graphs or impair independent movement of the hip and knee joints. Finally, 10 different topologies each having the desired properties can be derived. Figure 5.5 shows a choice of *morphological* representations for each of these topologies.

Even though we restrict ourselves in this study to these ten topologies, there are still many different *morphologies* which can be realized. The lengths of the wearable robot segments as well as the exact placement of the attachment locations determines the final structure of the wearable robot and can significantly alter the behavior of the structure.

5.2.2 Actuator placement

Apart from the choice of segment lengths and attachment placement for the wearable robot, there is another morphological choice that needs to be made. From figure 5.5 it can be seen that there are now 6 joints (3 attachments and 3 wearable robot) where potential actuators can be placed. Since we only have two degrees of freedom in this system, we should only need to actuate two of these 6 joints. We can however not actuate just any two of these 6, since not all combinations of two joints result in full control of the system. This is obvious when looking at the first topology, where we will need at least one actuator on the top chain (for the hip), and one on the bottom (for the knee). However, this is much less intuitive for the other topologies.

The possible actuator combinations for a given topology can be determined systematically, in particular since we only have a small number of joints. First, iterate all possible combinations of two actuator placements, i.e.:

$$\{(i, j), i \in [1, n], j \in [i+1, n]\}, \tag{5.5}$$

where $n = 6$ in our case. Then, for each actuator pair (i, j) we perform a degeneracy test on the topology such that if we *collapse* the two segments which a joint connects onto a single segment, we lose exactly one degree of freedom in the system. If this is the case for both i and j, then it follows that the pair (i, j) fully determines all of the degrees of freedom the system.

Figure 5.6 illustrates this procedure schematically for the verifying of the actuator pairs $(1, 4)$ and $(1, 6)$ for topology 2. For both, first joint 1 is collapsed, resulting in $k = 1$, i.e. the loss of

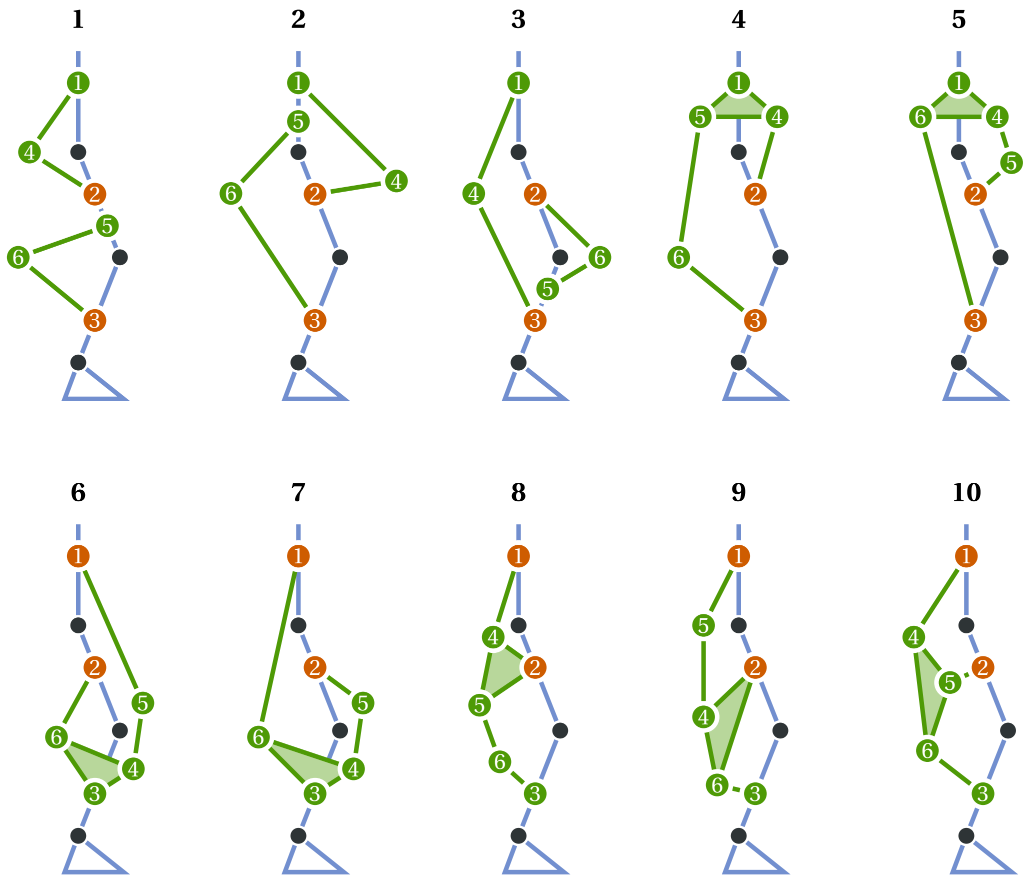


Figure 5.5 – Morphological representations of enumerated topologies having 3 wearable robot joints and 4 wearable robot segments. Only a single realization for each topology is shown, and it should be noted that different realizations of the same topology can lead to wearable robots with significantly different properties. Joints in green (1, 4, 5 and 6 in the top row, and 3, 4, 5 and 6 in the bottom row) are the added attachment and wearable robot joints, while the orange joints (2 and 3 in the top row, 1 and 2 in the bottom row) are the added loop closure joints.

one degree. Next, for the first pair joint 4 is collapsed, however the degrees of freedom in the system is still $k = 1$. The pair $(1,4)$ is therefore not a valid actuator pair. On the other hand, when collapsing joint 6 after joint 1 the result degrees of freedom are $k = 0$, and $(1,6)$ thus fully determines the system. Note that the maximum number of combinations of choosing joint pairs of size k from a total of n joints is given by:

$$N = \frac{n!}{2 \cdot (n-k)!} \tag{5.6}$$

(with $(i, j) = (j, i)$ and $j \neq i$), which means that in our case with $k = 2$ and $n = 6$, the maximum possible actuator pairs is $N = 15$. Table 5.1 lists the resulting actuator pairs for each topology.

In terms of structural parameter spaces, as defined when using MMPSO (see section 3.2), we

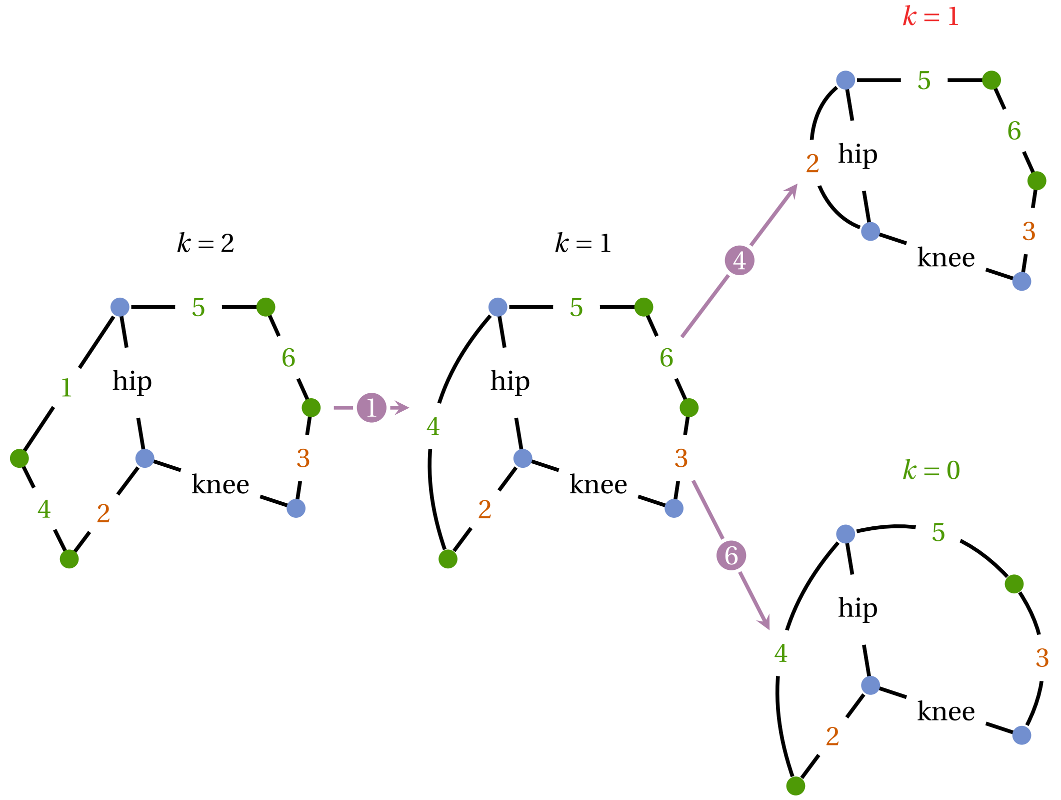


Figure 5.6 – Schematic illustration of the determination of valid actuator pairs in given a topology. In this case, the pairs $(1,4)$ and $(1,6)$ are checked for topology 2. We start on the left with the full topology and then collapse joint 1 to obtain the second topology and check for the mobility of the topology using equation 5.1. If the mobility has reduced by a single degree when collapsing a joint, then that joint can be a valid actuator. Next we check for the second actuator in each pair. When collapsing joint 4 (top right), we obtain $k = 1$ and thus DOFs are not reduced and the pair is invalid. On the other hand, collapsing joint 6 (bottom right) results in $k = 0$ and thus $(1,6)$ is a valid actuator pair.

now have 10 parameter spaces for the morphological parameters needed to realize the 10 topologies (i.e. wearable robot segments and attachment offsets). In addition, within each of these 10 spaces, we have a further N parameter spaces for each of the actuator pairs. We therefore have a total of 126 parameter configuration spaces to explore using MMPSO. Section 5.4.1 provides more details on using MMPSO for this particular optimization.

5.2.3 Singularities

Although the general, or maximum mobility of a closed loop kinematic chain can be determined using equation 5.1, the actual mobility of a parallel system is usually *variable*. The cause of this variability is the occurrence of kinematic singularities, in which case the mobility of the structure is temporarily reduced. When considering parallel structures such as shown

Table 5.1 – Actuator pairs for each topology

Topology	N	Actuator pairs
1	9	(1,5), (1,3), (1,6), (2,5), (2,3), (2,6), (5,4), (3,4), (4,6)
2	12	(1,5), (1,3), (1,6), (5,2), (5,3), (5,4), (5,6), (2,3), (2,6), (3,4), (3,6), (4,6)
3	12	(1,2), (1,3), (1,5), (1,4), (1,6), (2,3), (2,4), (3,5), (3,4), (3,6), (5,4), (4,6)
4	12	(1,3), (1,5), (1,6), (2,3), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6), (5,6)
5	15	(1,2), (1,3), (1,6), (1,4), (1,5), (2,3), (2,6), (2,4), (2,5), (3,6), (3,4), (3,5), (6,4), (6,5), (4,5)
6	12	(1,2), (1,3), (1,5), (1,6), (1,4), (2,5), (2,4), (3,5), (3,4), (5,6), (5,4), (6,4)
7	15	(1,2), (1,3), (1,6), (1,5), (1,4), (2,3), (2,6), (2,5), (2,4), (3,6), (3,5), (3,4), (6,5), (6,4), (5,4)
8	12	(1,3), (1,5), (1,6), (2,3), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6), (5,6)
9	12	(1,2), (1,3), (1,5), (1,6), (1,4), (2,5), (2,4), (3,5), (3,4), (5,6), (5,4), (6,4)
10	15	(1,2), (1,3), (1,4), (1,5), (1,6), (2,3), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6), (5,6)

in figure 5.5, kinematic singularities can easily occur. Figure 5.7 shows a simple case of such a singularity where in a particular configuration, a joint which could previously actuate the system can now no longer do so. The case shown in figure 5.7 is intuitive, but as with determining pairs of actuators, it is not necessarily so in the general case.

It is possible to exploit these singular configurations during locomotion since joints can effectively be locked. This can be beneficial during the stance phase since no torques are required to keep the leg straight. In earlier work, we would allow singular configurations to occur and we would indeed observe that most optimizations resulted in the use of singularities during the stance phase.

The problem is though that once in a singular configuration, we cannot move out of it by controlling the joint anymore. This problem does not occur during simulation as such because we *optimize* for steady state locomotion and we do not explicitly require the system to be controllable everywhere as long as it moves forward. Therefore, as long as the structure is no longer singular when we need to actually control it (e.g. end of stance), gaits are unaffected. Of course, on a real device the type of singularities that would occur with our wearable robot are highly undesirable. Unless additional mechanisms are present to discontinue the singular configuration at any time, the system becomes uncontrollable during certain phases of the gait and it is no longer possible to obtain anything other than the precise gait we optimized for.

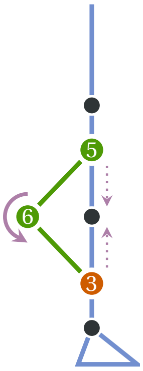


Figure 5.7 – Occurrence of a simple kinematic singularity. It is intuitive in this case that applying a torque on joint 6 would result in only linear forces acting on the knee joint, and not in any torque. Therefore, no amount of torque applied to joint 6 will result in any movement of the knee, hence the singularity. Note that a singularity is not a property of the kinematic chain only, it depends on where torque is being applied in the chain. In this case, the knee joint itself (for example) can still be actuated without problems.

Until now, we have only mentioned singularities in the context of the wearable robot. However, singularities can also occur on the human joints, i.e. leaving the person locked in the wearable robot structure. Again, this is undesirable since not only does this lead to possible discomfort for the person, but he or she should always be capable of control.

In our simulations we therefore explicitly validate morphologies in terms of possible singularities before simulating them. Since we are interested in steady state locomotion, we look at singularities occurring during nominal walking gait trajectories. Furthermore, we verify that neither the actuated joints on the wearable robot, nor the human hip and knee joints become singular during any phase of the gait. Determining the singularity of a configuration is detailed in section 5.4.4.

5.2.4 Summary

In summary, the goal of this work is the co-design of a wearable robot designed for lower limb locomotion of an average, adult size human person. We focus specifically on steady state locomotion in the sagittal plane by attaching a parallel structure to the hip and knee of a human model, while leaving the ankle actuated directly. The morphological space of the wearable robot consists of the realization of 10 different topologies featuring two degrees of freedom per leg, composed of 4 additional segments and 6 additional joints. These two degrees of freedom are both actuated using wearable robot joints and the choice of actuator pair is open for optimization, while the remaining wearable robot joints are modeled as spring dampers (of which spring stiffness and damping are also optimized). Finally, we explicitly verify that a wearable robot does not contain kinematic singularities during nominal gait before simulating it, for both the actuated wearable robot joints as well as the human hip and knee joints.

5.3 Modeling

There are three main difficulties when it comes to the modeling of wearable robot structures for locomotion as we would like to do here. The first is the modeling of closed loop dynamics, for which the resulting equations of motion are significantly harder to derive than for the general open loop system. Moreover, we also need a convenient representation of the model in which we can easily parametrize not only the control, but the morphology as well, without loss of performance. The second issue is that of simulating contacts with the environment. Although this is a general issue when modeling for locomotion simulations, it is even more so when considering hard contacts with closed loop dynamics. Finally, since we would like these simulations to provide input to the design process of an actual wearable robot, we need simulations to be accurate (rather than very fast) and have access to quantities such as joint constraint forces and model Jacobians (used to check for singularities).

Although it is certainly possible to perform the type of simulations that we require using various simulator tools, cȯdγn (see chapter 2) is particularly well suited for the task. Not only does it provide a hard contact model which solves for the dynamics regardless of closed loops present in the system, it also allows for both easy parametrization of the morphology and high performance simulations. We therefore use cȯdγn for all of the simulations in this chapter. We briefly describe in the next section the modeling of the human augmented with the wearable robot and continue after that with the method used to verify the singularity of the system.

5.3.1 Augmented human model

The modeling of the human is done in exactly the same way as shown in the previous chapter (section 4.1.1), except that we no longer use the soft contact model. The dimensions and inertial properties are kept the same, i.e. resembling an adult sized human of approximately 70 kg and a height of 1.80 m.

Since we limit ourselves to 10 topologies, we can construct the closed loop rigid body dynamics for each of these topologies separately. By choosing the proper parametrization, we can then still use cȯdγn to generate high performance code to forward simulate the system. Recall from chapter 2 that cȯdγn does not allow dynamic alteration of the structure of the dynamics. It is therefore less suited if we would like to also optimize for open topologies (e.g. using genetic programming), since we would need to reconstruct the dynamics for each simulation. Here instead, we can parametrize each morphology (corresponding to one of the 10 topologies) such that we can still change mass, center of mass, inertia, segment length and attachment positioning while avoiding the need to regenerate the model for each simulation.

Having defined the human model, we then proceed to first place the three attachment joints on the torso, upper leg and lower leg of the human. Then we continue to introduce the four additional wearable robot joints and connect them according to the respective topology.

5.3.2 Closing joint actuation

In section 2.6.9 of chapter 2 the procedure for deriving the closed loop equations of motion was shown. This derivation only considered the kinematic constraint imposed by the loop closure joint, and omitted how to model the application of active joint torques on the loop closure joint. However, since we are able to place actuators on loop closure joints, we require additional dynamics integrating active loop closure forces.

The easiest way to introduce these forces, from Featherstone (2008), is to model the loop closure joint forces as external forces projected to the bodies connected by the joint. Instead of calculating $\boldsymbol{\tau}_a$ explicitly, we can instead calculate $\boldsymbol{C} - \boldsymbol{\tau}_a$ directly by modifying the Recursive Newton Euler algorithm as shown in section 2.6.6. This is automatically done in cȯdγn as soon as there are loop closure joints.

5.3.3 Constraint forces in closed loop systems

Since we derive equations of motion in generalized coordinates, we no longer have access to the constraint forces from the equations of motion. This is not important when concerned only with motion, but becomes important when using simulations for design. In particular, we are interested here in being able to measure constraint forces (i.e. internal forces) on the human joints and segments.

Using the Recursive Newton Euler algorithm, constraint forces can actually be obtained from a by-product of the derivation of $\boldsymbol{C}$, and thus requires no further calculations (see equation 2.53). This however is no longer sufficient when considering closed loop systems. Recall that loop closures are resolved in the equations of motion by introducing acceleration constraints:

$$\boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k} \tag{5.7}$$

and generalized constraint forces:

$$\boldsymbol{\tau}_c = \boldsymbol{K}^T \boldsymbol{\lambda}, \tag{5.8}$$

where $\boldsymbol{\lambda}$ are the loop closure constraint forces as observed from the child frame of the loop closure joint. Note further that although $\boldsymbol{\lambda}$ is solved for, it is merely projected directly to generalized forces $\boldsymbol{\tau}_c$ and thus information about constraint forces is again lost. To resolve this, we can project $\boldsymbol{\lambda}$ through the constraint force subspace of the loop closure joint to obtain the spatial constraint force at the joint and propagate it through the system as an external force. We only do so to compute the resulting constraint forces, i.e. compute them separately from the equations of motion. Incidentally, since the hard contact model in còdγn is implemented using loop closure constraints, this method also works without changes for obtaining joint constraint forces when using hard contact models.

5.3.4 Control

For the control of the wearable robot joints we use the same impedance controller developed in the previous chapter. There we observed that the *step* control mode, i.e. where stiffness and damping are varied by a step function, generally outperforms a constant stiffness/damping impedance controller, as well as a more complex stiffness/damping controller (the *ppoly* controller). It therefore seems that it provides a good trade-off between parameter complexity and variable impedance control during optimization. Furthermore, using impedance control we obtained natural gait from high-level objectives only. We will therefore use the *step* controller in this work to control the wearable robot:

$$\tau_i(t) = K_i(t)(q_{r_i}(t) - q(t)) - D_i(t)\dot{q}(t) \tag{5.9}$$

$$K_i(t) = \begin{cases} K_i^1, & K_{t_1} < t < K_{t_2} \\ K_i^2 & \text{otherwise} \end{cases} \tag{5.10}$$

$$D_i(t) = \begin{cases} D_i^1, & D_{t_1} < t < D_{t_2} \\ D_i^2 & \text{otherwise} \end{cases}, \tag{5.11}$$

with $\tau_i(t)$ the output torque, $q_r(t)$ the joint angle reference trajectory, $K_i(t)$ the phase dependent stiffness and $D_i(t)$ the phase dependent damping. It should be noted that here we did not explore the use of other possibly interesting control methods. We also do not explicitly investigate the role of variable impedance for actuation of the wearable robot. Our main focus is on the development of the methodology and we build on evidence from our previous results towards variable impedance control being suitable for human locomotion optimization.

The joint angle trajectories of the wearable robot joints can be more complex, or variable, due to the non-linearity of the joint angle transfer function of the parallel structure. We therefore increase the number of points for the piecewise polynomial function for $q_r(t)$ to 5 (instead of 4 previously).

Apart from the actuated wearable robot joints, the ankle joint is also actuated using the same type of *step* controller. The remaining wearable robot joints are furthermore controlled using a simple spring/damper controller:

$$\tau_i = K_i(q_{r_i} - q) - D\dot{q}, \tag{5.12}$$

of which the stiffness K_i, rest angle q_{r_i} (constant) and damping D_i are open for optimization.

5.4 Optimization

The optimization procedure for the co-design of the morphology of the wearable robot and its control deserves some special consideration. On the one hand we do not only optimize for a fixed set of numerical parameters, but are required to optimize for structurally different sets of parameters (i.e. for the different topologies and actuator positioning). On the other hand, we know beforehand which types of solutions we would like to explore, and those solutions can be easily enumerated (i.e. 10 topologies with actuator pairs varying from 9 to 15). It is therefore neither completely fixed, nor completely open-ended.

5.4.1 Algorithm

The Metamorphic Particle Swarm Optimization algorithm developed in chapter 3 is particularly well suited for this type of optimization, and in fact was specifically designed for it. It allows for guided exploration of a fixed number of known parameter spaces, using cooperation principles to transfer particles between parameter spaces akin to the original Particle Swarm Optimization method.

In the parlance of MMPSO, the wearable robot co-design optimization has a single parameter *pool* containing 126 parameter *groups*. Each group is mutually exclusive within the *pool*, so only a single topology and actuator pair can be active at the same time. A second parameter *pool* with a single parameter *group* is used to contain all of the common parameters to be optimized (such as for the ankle joint actuation and wearable robot segment lengths). Each of the 126 parameter *groups* corresponding to a particular topology and actuator pair contains the parameters for the active joints in that *group*.

5.4.2 Parameterization

There are different ways in which we can parametrize the wearable robot morphology and control. For the control we choose the same parametrization as we have used in the previous chapter, i.e. a differential encoding of the trajectory signals. For the morphology we can choose between two different encodings. The first encodes for each set of connected links the angle between the links, and for each individual link, its length. The second approach simply encodes, for each joint, its position in Cartesian space, from which the segment lengths can then be derived by means of looking at the connectivity of joints according to the topology. Both encodes require the same number of parameters (e.g. 4 parameters for topology 1 and 6 parameters for topology 5).

An important difference between the two encodings is the manner in which a change in a parameter results in a change in the morphology. Using the first encoding, a change in a single parameter, for example a joint angle, causes a rotation of a (sub)structure of the morphology. On the other hand, using the second encoding, a single parameter change simply results in

Table 5.2 – List of open parameters

Parameter	Boundaries			Description
				Morphological parameters
Body joint				
x	$(-0.15,$	$0.15)$	m	The Cartesian x-position of the joint
z	$(0.1,$	$1.3)$	m	The relative z-position of the joint on the segment at which it is placed
WR joint				
z_{rel}	$(0,$	$1)$		The Cartesian z-position of the joint
				Control parameters
Active joint				
$q_{x_1 \ldots x_6}$	$(0,$	$1)$		Differential x points for reference position signal q
$q_{y_1 \ldots y_5}$	$(-\pi,$	$\pi)$	rad	y data points for reference position signal q
K_{x_1,x_2}	$(0,$	$1)$		Differential encoding of stiffness step transition
K_{y_1,y_2}	$(0,$	$12000)$	$\mathrm{N\,m\,rad^{-1}}$	Active impedance stiffness
D_{x_1,x_2}	$(0,$	$1)$		Differential encoding of damping step transition
D_{y_1,y_2}	$(0,$	$60)$	$\mathrm{N\,m\,s\,rad^{-1}}$	Active impedance damping

a global Cartesian displacement. It is not entirely intuitive to say beforehand which type of encoding is better than the other. Here we chose to use Cartesian placement of joints since parameters encoding for morphology are more decoupled in a sense. The Cartesian placement is done on the human model in the upright position, and is therefore equal for the right and left leg.

For joints which are attached to the human body, we use a relative displacement encoding, i.e. a value from 0 to 1, encoding where on the corresponding segment between the parent and child joint the attachment is placed. For the wearable robot joints which are not directly attached to the body, we use a global Cartesian x and z encoding, allowing joints to be placed in a bounding rectangle around the human body.

Table 5.2 lists all the open parameters optimized in this work.

Mass distribution

The mass distribution of the wearable robot is automatically derived from its morphological configuration and the placement of actuators. We use reasonable estimates for the inertial properties of the actuators, passive spring/dampers and wearable robot segments, but at this time do not model these components precisely based on available physical devices.

Each actuator in the system is modeled as an object with a mass of 2 kg and a cylindrical moment of inertia around the axis of rotation for a cylinder with a radius of 5 cm. Similarly, the passive joints are modeled using a weight of 0.5 kg and also have a cylindrical moment of inertia. The wearable robot segments are approximately modeled after cross-sectioned aluminium bars with a thickness of 1 cm and a width of 3.5 cm, resulting in a density of $\approx 0.7\,\mathrm{kg\,m^{-1}}$ segment length. After wearable robot joints are placed, resultant mass m, center of mass CoM and inertia I for each body is determined by composition of the individual inertial properties belonging to the same joint. Triangular segments (featured in topologies 4 to 10) are modeled using three segment bars.

5.4.3 Objectives

Since MMPSO can run any type of optimization algorithm in the *inner* layer, we can readily use particle swarm optimization with lexicographic ordering in the *inner layer*. We use the same structure for the lexicographic objectives as used in the previous chapter, with a few modifications. Two new objectives are added to the previously used objectives. The first is an explicit check on segment size. Since the parametrization does not explicitly encode for segment length, the resulting segments can become very long in certain cases. We therefore introduce an objective which verifies if segments are not longer than a certain size. This maximum size is set to 0.7 m in this work, based on approximate maximum length at which the segments would start to buckle according to their modeled material. The second new objective is one verifying that the kinematic configuration does not become singular, and is explained in greater detail in the next section.

When we initially ran simulations, we noticed that it was difficult to optimize for *speed* and *assist time* in sequence. An improvement in *assist time* would directly have a negative impact on the *speed*, thus oscillating between the two objectives. This is one of the possible disadvantages of lexicographic ordering of objectives. To resolve this issue we instead combine the two objectives into a single lexicographic objective by using a multiplicative aggregation function. Note that the design of an objective function is still an empirically guided effort, which often requires several iterations of tuning before desired results can be obtained. Since we found it significantly more difficult to obtain gaits which were self-stabilizing, we also allow a maximum assist time of 10% during the gait. Finally, we use power (i.e. torque times angular velocity) instead of simply torque. We do so because due to the non-linear nature of the transfer function relating the joint velocities of the wearable robot and the human joints, we no longer assume that torque is a reasonable estimate of energy. If we were to use torque instead, we would bias towards possibly fast moving wearable robot joints requiring relatively small torques. The final objectives are listed in table 5.3.

Table 5.3 – Lexicographic Objectives

Objective	Until		
1. time	time	=	max time
2. segment size	segment size	≤	max segment size
3. singularity	singularity	=	0
4. speed match · no assist time	speed match	≥	0.95 & assist time ≤ 0.1
5. -power	∞		

5.4.4 Singularities

There are two types of singularities that we want to avoid during the co-design simulations. The first type is caused by not being able to reach a certain hip/knee joint angle combination due to the wearable parallel structure not allowing to reach that far. This effectively creates unidirectional singular configurations, and often happens for example in serial manipulators when trying to reach for a target Cartesian position which is out of reach. In our case this would mean that some hip/knee angles could not be reached during nominal gait, which is undesirable. The second type of singularity is the one discussed earlier, where there is a loss of mobility. In these cases it is no longer possible to control the system. Additionally, we require that the human is always able to control the system, and thus the hip and knee joints should also never become singular.

The aim of the singularity verification procedure is to quickly discard solutions which are not interesting, without the need to simulate them extensively. We begin by obtaining the nominal joint angle trajectories during normal walking from Winter (2009). We then discretize these trajectories such that we obtain N pairs of (hip, knee) joint angles. We then verify both types of singularities for each pair of (hip, knee) angles, given the wearable robot as given by the parameters of the solution to be evaluated. Furthermore, we also evaluate singularities for $\pm\Delta^\circ$ offsets at each point to make sure there are no singularities in the neighborhood of the nominal gait cycle.

The determination of the first type of singularity is closely related to the problem of inverse kinematics. Given (hip, knee) joint angles, we want to determine the dependent joint angles of the wearable robot (since we only have two degrees of freedom), given the loop closure constraints. If the conditions for the determination of these dependent joint angles are ill, then the configuration has become singular. There are various ways in which to solve for the dependent joint angles. If a closed form solution of the system is known (i.e. a function mapping independent joint angles to dependent joint angles), then such conditions can be readily obtained from the closed form equations (i.e. in the case of singularities there will be no solution). However, as mentioned in chapter 2 when discussing closed loop systems, it is hard to derive closed form solutions in general, and it is not the approach we are taking.

If a closed form solution is not available, an iterative approach can be used instead. A fundamental quantity that enables this is the Jacobian. Recall from section 2.6.8 that the Jacobian can be used to map joint velocities to Cartesian velocities, i.e.:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}} \tag{5.13}$$

Therefore, given a (small) error Δx in location of an attachment point on the human body, and the end-effector of the closing loop joint (i.e. the loop closure error), we can obtain joint velocities that would reduce this error by inverting this equation:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\boldsymbol{\Delta x} \tag{5.14}$$

Of course, $\boldsymbol{J}$ is linearized, and thus the relationship between $\dot{\boldsymbol{q}}$ and $\dot{\boldsymbol{x}}$ is only approximated linearly. Therefore, care has to be taken when the error Δx is large. This is usually solved by performing equation 5.14 iteratively, reducing Δx over a number of steps. Furthermore, $\boldsymbol{J}$ is not always invertible in which case the pseudo inverse $\boldsymbol{J}^+$ can be used instead.

Since cȯdyn readily provides the required Jacobians, we can easily perform this iterative Jacobian approach for solving the inverse kinematics. To detect singularities, we simply observe Δx and if we are unable to reduce it until some (small) threshold, we decide that the configuration is singular. The same procedure is also used to determine the initial conditions of the wearable robot given some initial condition for the human joints, i.e. initialized inside the gait.

If a singularity is not reached when closing the loop closure joints on the desired (hip, knee) configuration we can proceed with verifying the second type of singularity. Again, the Jacobian comes to our rescue. The principal idea is that we need to verify if applying a velocity to an actuated joint would result in a non-zero velocity on desired joints (in our case those of the human), while not violating the kinematic constraints imposed by the loop closure joints. To do so we first construct the loop closure Jacobians:

$$\dot{\boldsymbol{x}}_c = \boldsymbol{J}_c\dot{\boldsymbol{q}}, \tag{5.15}$$

for each loop closure joint c, mapping joint velocities $\dot{\boldsymbol{q}}$ to a Cartesian velocity $\dot{\boldsymbol{x}}_c$ in the coordinate frame of joint c. We then create an extended Jacobian by stacking the individual Jacobians:

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_1 \\ \vdots \\ \boldsymbol{J}_N \end{bmatrix} \tag{5.16}$$

Finally, we can create a mapping from one joint velocity to another under the kinematic constraint from the loop closure joints by taking the *null* space projection of the Jacobian, i.e.

a matrix $\mathbb{N}$ such that:

$$\boldsymbol{J}\mathbb{N}\dot{\boldsymbol{q}} = 0 \tag{5.17}$$

In other words, $\mathbb{N}$ maps joint velocities such that they do not affect the loop closure constraints, and $\mathbb{N}$ is obtained by:

$$\mathbb{N} = \boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J} \tag{5.18}$$

Having obtained $\mathbb{N}$, the occurrence of singularities can be easily verified. Given a actuated joint i and an observable joint j, we can simply verify whether $\mathbb{N}_{(i,j)}$ is non-zero. If so, it means that a velocity in i results in a velocity in j (and vice versa) by effect of the loop closure constraint. If $\mathbb{N}_{(i,j)}$ is close to zero, we decide that the configuration is singular in i with respect to j. We then verify this for 1) the actuated wearable robot joints towards the hip and knee joints, and 2) the hip and knee joints towards themselves. The second check ensures that the human is always able to control the motion of the robot.

There is one snag related to determining the singularity of the loop closure joints themselves. Since they are not represented by a generalized coordinate, we can not use $\mathbb{N}$ to determine the influence of a velocity on a loop closure joint to other joints in the system. There are two ways in which this can be solved. The first is to create new systems in which a different joint becomes the loop closure joint, leaving us free to use the $\mathbb{N}$ of that system to determine singularity of the original closure joint. This however requires the definition of multiple models for each topology which is rather inconvenient. The other solution, which is the one we use here, is to use the loop closure velocity Jacobian. Recall that the loop joint spatial velocity is defined by:

$$\boldsymbol{v}_J^c = (\boldsymbol{J}_i - \boldsymbol{J}_j)\dot{\boldsymbol{q}}, \tag{5.19}$$

where $\boldsymbol{J}_i$ and $\boldsymbol{J}_j$ are respectively the Jacobians of the parent and child bodies of the loop closure joint. Thus, given a mapping of loop closure joint generalized velocity $\dot{q}_c$ to its spatial velocity $\boldsymbol{v}_J^c$ (given by the motion subspace of the loop closure joint), we can use the inverse of the loop closure velocity Jacobian to obtain how the spatial velocity $\boldsymbol{v}_J^c$ affects the generalized velocities of the system $\dot{\boldsymbol{q}}$:

$$\dot{\boldsymbol{q}} = (\boldsymbol{J}_i - \boldsymbol{J}_j)^{+}\boldsymbol{S}_c\dot{q}_c, \tag{5.20}$$

with $\boldsymbol{S}_c$ the motion subspace of loop closure joint c. We can then simply check whether a velocity in an actuated loop closure joint affects the hip and/or knee joint, thus verifying whether or not the loop closure joint is singular. The overall procedure for verifying that a given configuration is not singular for a range of (hip, knee) joint angles is then as follows:

1. Use the loop closure joint end-effector Jacobian (pseudo)inverse to iteratively solve for

initial conditions of the (hip, knee) joint angle pair

2. If the loop closure can not be closed in some iterations, decide the configuration is singular
3. Compute the null space $\mathbb{N}$ of the extended Jacobian containing all loop closure joint Jacobians
4. Determine if the knee or the hip joint is singular towards itself, if so the configuration is singular
5. For actuated wearable robot joints determine whether they are singular towards the knee and hip joints by using $\mathbb{N}$ in case of non loop closure joints and equation 5.20 otherwise. If singular in either hip or knee, then the configuration is determined singular

This verification is performed each time before simulation starts, and solution which are singular are discarded directly without further simulation.

5.5 Results

We run the MMPSO optimization with 200 particles, since there are many possible configurations to explore. Furthermore, we run it for 800 iterations thus allowing time to properly explore configurations as necessary, before converging on a single configuration and finishing the optimization. The probabilities P_e, P_l and P_g were chosen as shown in figure 5.8. The probabilities are chosen empirically, however not arbitrarily. Since we explore a large space, we allow ample time for parameter subspaces to be explored through the exploration probability p_e. We then allow, through P_l, more intense exploration of particles' local best known configuration subspaces. Finally, P_g becomes active around iteration 500 and ensures particles converge on a single global best known parameter subspace and spend the last 200 iterations fully exploring the solutions found there.

Table 5.4 – Top 3 obtained power solutions

Topo	**Mass**	**CoM (z)**	**Segment size**	**Power**	**Torque**	**Assist time**
2	83.3 (+13.3)	1.07 (-0.03)	0.3–0.46	462	1141	0
5	83.9 (+13.9)	1.02 (-0.08)	0.2–0.52	748	418	0.01
3	83.5 (+13.5)	1.03 (-0.07)	0.1–0.64	1018	818	0

Table 5.4 lists various quantities of the 3 best obtained solutions in terms of power. As can be seen, optimizing for power leads to different optima as when optimizing for torque, since the second ranked solution for power would rank first when comparing only torque.

Furthermore, what is interesting to notice is that although each solution has optimized for a different topology, the added mass (≈ 13.5kg) as well as the total center of mass of the system is approximately the same. Note that a fixed amount of mass of 11 kg is already added in all configurations due to the 4 actuators (each 2 kg) and 6 passive revolute joints (each 0.5 kg). The

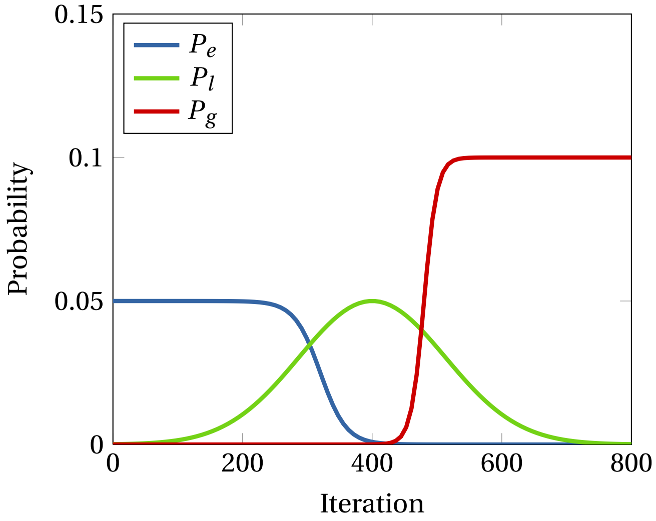


Figure 5.8 – Probability curves determining the rate at which particles are mutated to a different parameter subspace. The curves are designed such that there is early exploration and late convergence towards a global optimum. Since the optimization problem is complex, particles mutate on average only every 20 iterations, as determined by P_e in the beginning. From iteration 500, particles start to transfer to the globally best known parameter subspace by P_g.

center of mass, when compared to the non-augmented human model, is moved only slightly downwards, a maximum of 10 cm. What is particularly interesting though is that the center of mass has only changed in the z (up/down axis) direction. The wearable robot is otherwise perfectly balanced in terms of mass between the front and back sides, which was not explicitly optimized for. This can be explained by the fact that balancing the system as such improves stabilization of the system with minimum power requirements.

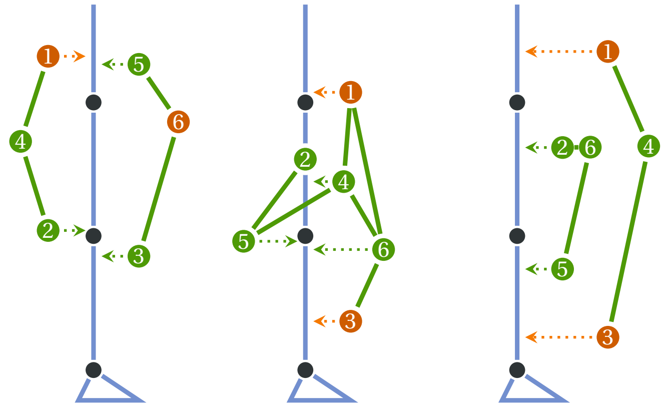


Figure 5.9 – Morphology of the 3 best obtained results. From left to right, topology 2, topology 5 and topology 3. Note that the wearable robot segments are offset in this schematic view from their original position, for clarity.

There are some further interesting observations to be made from the obtained solutions. In the first, the wearable robot joints 2 and 3 are placed very closely to the knee, and the joints 1 and 5 are placed relatively close to the hip. As such, the obtained morphology resembles that of an anthropomorphic structure, in terms of transfer of forces. Interestingly however, actuators are both placed relatively closely to the hip, thus concentrating most of the wearable robot mass at the level of the hip.

The second morphology features a complex linkage system which operates in an accordion like manner, expanding during swing phase and contracting closer to the body in the stance phase. Interestingly, the wearable robot joint 1 and joint 3 perform actuation of the system, while the relatively complex linkage system composed of joints 2, 4, 5 and 6 provides transfer of the appropriate movement to the human joints. The gait of this second solution is somewhat less natural. It makes fairly large steps, which is likely the cause of it performing worse (in terms of power) than the first solution. The last morphology is in a way quite similar to the second morphology, since again joints 1 and 3 actuate the system while the internal linkage system between the upper leg and the lower leg is completely passive. However the resulting gait is optimized as to keep the knee straight at all times, leading to a relatively unnatural gait. Figures B.3 and B.4 in appendix B show snapshots of the gaits of the first two solutions.

Next we can look at the behavior of the MMPSO optimization algorithm. In particular, we are interested in how well MMPSO has explored the parameter configuration spaces. Figure 5.10 shows a visiting frequency plot for the first two best obtained solutions (the third shows the same characteristics). We can immediately see that most of the parameter subspaces are visited, except for 4 spaces which are never visited. On average, ≈ 700 evaluations are performed in each subspace, while the optimal subspace has seen ≈ 61700 evaluations (note that the scale in the figure is a $\log_2$ scale). This large difference is due to the global exploitation probability P_g which we designed such that the last 200 iterations were mostly spend in a single subspace.

Not all parameter subspaces are explored equally, which is due to the stochastic nature of our optimization algorithm. It is therefore always important to run an optimization several times with different initial conditions. Additionally, the algorithm does not guarantee visiting all subspaces, as can be seen in figure 5.10. This is also not the purpose of MMPSO, since in that case it would be more appropriate to simply do a systematic search. Instead, MMPSO is designed such that it rapidly explores multiple possible spaces, while making informed decisions on which of those spaces are most interesting to explore further.

Videos of the resulting wearable robot solutions can be found at http://thesis.codyn.net/videos/codesign/last.

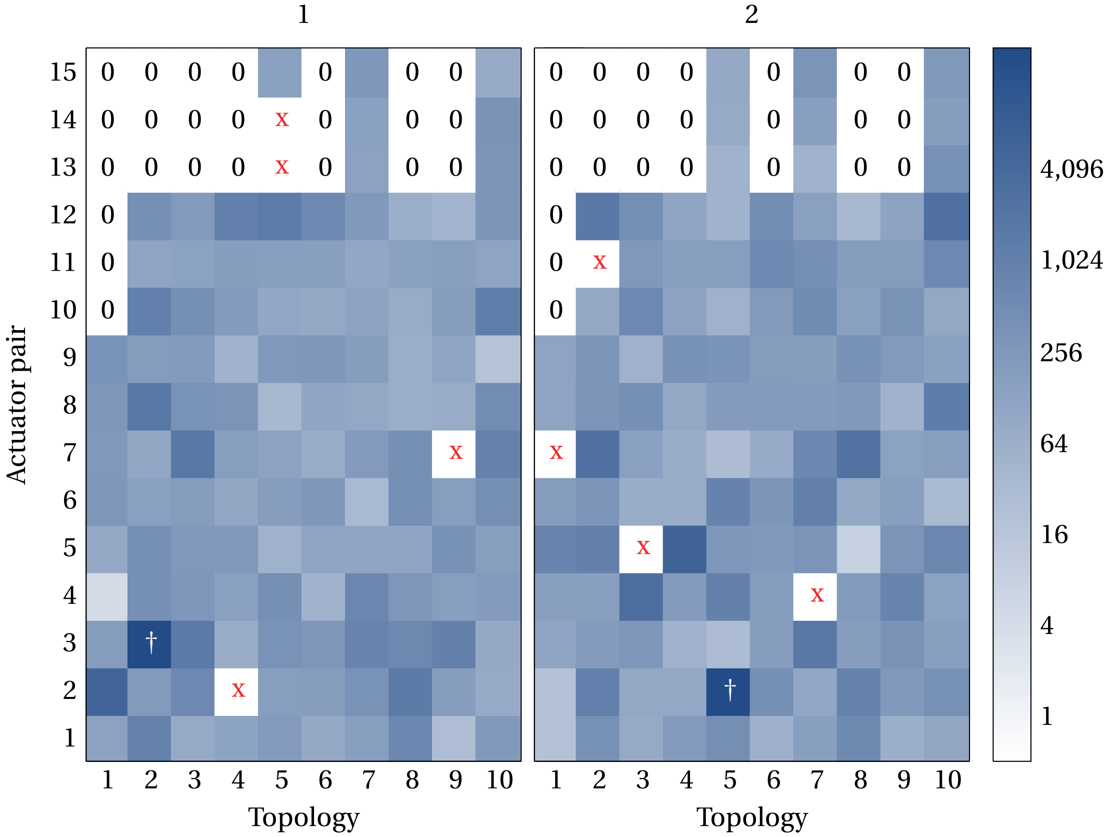


Figure 5.10 – Number of evaluations in each parameter subspace. Each topology (on the x axis) corresponds to the topologies shown in figure 5.5, while each actuator pair corresponds to the pairs listed in 5.1. Cells indicated with 0 are invalid configuration spaces (i.e. topology 1 has only 9 possible actuator pairs). Furthermore, cells indicated with x are not visited at all while the cell marked with † indicates the topology and actuator pair of the optimal solution found in that particular run.

5.6 Discussion

In this chapter we used all methods previously developed: 1) closed loop rigid body dynamics and efficient model parametrization from chapter 2; 2) simultaneous optimization of known sets of parameter configuration spaces and their continuous parameters, as well as large scale population based optimization from chapter 3; and 3) the impedance control and lexicographic ordering method for multiobjective optimization from chapter 4. We show how all these combined can be used to derive a methodology for the co-design of the structure and control of a wearable, lower extremities robot to effectively explore the design of a non-anthropomorphic augmenting structure for human locomotion assistance.

The open-ended exploration of the morphology and control of such a device leads to a complex optimization problem with many possible parameter configuration spaces and, within each, a large space of continuous parameters to be optimized. Nevertheless, we show that our methodology is able to explore this design space and obtain wearable robot morphologies with varying dynamical properties. In particular, it is interesting to note how the mass distribution of the structure is optimized such as to have a minimal impact on the center of mass of the total system when compared to a human model, without having specified the need for it explicitly. Additionally, we see that all obtained solutions favor placing at least one actuator attached to the torso (just above the hip), which we presume to increase stability. The non-anthropomorphic structure is able to transfer motion to the hip and knee joints through a series of parallel structures which would not be possible using a conventional design.

In the results shown here, we do not obtain a more energy efficient design of a wearable robot than the anthropomorphic case. The obtained solutions all require more power than if we would simulate an anthropomorphic wearable robot with the same material properties (i.e. masses) as the ones used for the non-anthropomorphic wearable robot during our study. However we did not set out with this goal in mind. Instead, we show that our methodology enables the effective exploration of a possible solution space leading to novel and not always intuitive solutions. Our methodology furthermore allows one to easily optimize only parts of the resulting system when a certain interesting dynamical property of a structure is found, leading to an iterative co-design method. For example: 1) the morphology could be fixed, focusing optimization efforts on (possibly more complex) controllers only; 2) only actuator placement could be explored in a single topology; or 3) only a subchain of the non-anthropomorphic structure could be left open for further optimization.

In many ways, the results shown in this study only touch upon the subject of the co-design of a device as we propose. It should be noted that we do not aim to provide a finalized design. Rather we propose that our methodology is a viable solution for exploration of possible non-conventional designs. The methodology is general enough to also allow for the exploration of *conventional* (i.e. anthropomorphic) design, where it could be used to optimize for mass distribution, minor actuator placement adjustments or other morphological design parameters. Although we show that this method obtains reasonable locomotion gaits supporting the human body, exploring many possible morphologies and corresponding control of those, there are still many directions left to explore further.

1. *Interaction forces*: We presented in this chapter a method to obtain the interaction forces in closed loop systems (which is provided by cȯdγn currently), however we did not optimize specifically for these forces. Although the nature of the optimization is such that, implicitly, by optimizing for the power of the wearable robot, the interaction forces should be minimized. In other words, the wearable robot is more efficient the more power is transferred to the human joints, instead of ending up in constraint forces. Nevertheless, there are no guarantees towards this end and it would be interesting to add a lexicographic objective for a maximum allowed interaction force on the human limbs.
2. *Ankle support*: In our present study we only investigate the actuation of the hip and knee joints through the wearable robot, leaving the ankle conventionally actuated. However,

a realization of such a device would at the very least need a way to transfer load from the wearable robot to the ground, since the added mass from the device would add too much load on the human body. It would therefore be necessary to either 1) design an ankle device connected to the hip/knee device which transfers the load, or 2) co-design the actuation of the ankle as well.

3. *Passive elements*: The non-actuated joints in our study are completely passive. It is however well known that it is possible to store and release energy during certain phases of the gait, enabling more energy efficient devices. It would be interesting to explore the incorporation of such energy stores in the passive joints of the wearable robot during the co-design optimization process (Donelan *et al.*, 2008). Examples of possible avenues are the exploration of rotational spring dampers, clutches allowing for the temporary locking and unlocking of a structure (Herr, 2004) or the addition of linear elastic elements (van den Bogert, 2003).
4. *Human adaptation*: Our study is only concerned with the full actuation of the wearable robot, while the human inside is completely passive. This resembles more or less the case of a paraplegic, but does not correctly represent other possible uses for such a design. The adaptation of a human to a device can be taken into account while assisting only a certain amount of the full locomotion gait, as opposed to providing all of the necessary power (Ronsse *et al.*, 2011a). Furthermore, the role of morphology and control for various pathologies (physical limitations or unnatural motor control strategies) can be explored by modeling such pathologies in simulation (Delp *et al.*, 2007).
5. *Generalization*: Here we have only explored applying our methodology to a specific case. Although the methodology itself is general (i.e. the algorithms and methods do not assume domain specific information), the design of the objectives and choice of parametrization is still specific to the chosen problem. In Pouya *et al.* (2010) we show that the methodology can be applied to other domains, but this only gives emperical evidence towards generalization. As is usual with optimization, the obtained results are sensitive to the specific objectives that were optimized for. In our experiments we specifically optimized only for steady state locomotion in the sagittal plane. It is therefore not unreasonable to assume that the obtained morphologies and control are specific to this task, and do not necessarily generalize to a design suitable for a larger range of tasks. However, the method itself does not exclude the design of a process in which the objectives include a multitude of tasks (for example including sitting, standing up and walking stairs).

6 Conclusion

We started out looking at the possibility of the co-design of a lower extremities, wearable robot using evolutionary optimization strategies. After having performed various initial experiments we realized two important things, 1) there is great potential for novel and unintuitive design strategies arising from the co-design of morphology and control and 2) the simulation problem is a complex one and requires a state of the art and, importantly, *open* simulation environment.

We thus took a step back, and started working on a framework which resulted in cȯdyn, a state of the art modeling and simulation environment for the design of coupled dynamical systems with a specific focus on coupled oscillators and rigid body dynamics. Driven by the desire to create an open, free to use, fast and well documented framework, we created software in which modeling coupled dynamical systems is both expressive *and* well performing. Furthermore, unsatisfied with the currently available, state of the art RBD simulators suitable for accurate simulations including precise, hard contact models and sophisticated closed loop dynamics modeling, we provided a fully declarative implementation of RBD as described by Featherstone (2008). The resulting models are easy to parametrize and can be quickly translated, without loss of generality, to a low level implementation suitable for simulation on Real Time, embedded and micro-controller systems. By making the framework available under an open and free license at http://www.codyn.net, results obtained using cȯdyn can be readily replicated and improved upon since models are easily shared. It therefore provides a basis upon which scientific work can be rapidly advanced. cȯdyn is not suitable for all types of dynamical systems. In particular, it only supports systems which can be modeled with ordinary differential equations. It also does not support dynamical systems with fully variable dynamical structure which should change after the model has been constructed. These limitations do not affect the work presented here, but it means that cȯdyn is unsuitable for the modeling of certain systems, such as modular robots. Furthermore, although the available contact models are suitable for locomotion, they are limited to point contacts and more complex, multi-contact models (with varying geometries) are currently not available.

Similarly, although less novel, we also developed a framework for the purpose of large scale, population based optimizations in a multi-user environment, in which simulation tasks are automatically scheduled on available resources, such as a cluster. Easy to configure and ready to deploy, it has provided the requisite infrastructure for many scientific works which would otherwise have been more time consuming to accomplish. A variety of population based optimization methods are provided within this open and freely available framework, including Metamorphic Particle Swarm Optimization.

MMPSO is a Particle Swarm Optimization based algorithm specifically designed for solving problems for which different possible solution structures exist, each with their own (possibly overlapping) set of continuous parameters. The codesign of a wearable robot is an example of such a problem, where the structure space consists of the wearable robot topologies and actuator placement and the parameter space consists of wearable robot Cartesian posiition and control parameters. MMPSO uses cooperative strategies, similar to those of PSO, to transfer particles between parameter subspaces in a probabilistic manner. These probabilities can be chosen in an informed manner, based on the complexity of the problem, the number of total subspaces, the number of particles and the maximum number of iterations.

To validate our developed tools for the codesign of the wearable robot, we first started considering the occurrence of natural human gait using impedance control by optimizing for high-level objectives only. Even though we only optimized for a specific target speed and minimized for a measurement of energy using lexicographic ordering of the objectives, we reliably obtained walking gaits with various global human characteristics. This validated both our optimization method as well as our control method (i.e. impedance control) as a reasonable representation of *humanoid* actuation. To investigate the role of human morphology for the performance of locomotion, we then applied the exact same method to a model of the CoMan humanoid robot. Here we found that it is not enough for the humanoid robot to be biomimetic, but that care should be taken in the design of its morphology when looking at locomotion. In particular we found that the feet were fundamental in obtaining any kind of reasonable gait and that the location of the center of mass significantly affected locomotion performance. To show the role that *variable* impedance can have, we furthermore performed a perturbation study in simulation where we observed that optimizing only for stable walking, impedance was modulated such that periodic perturbations during the swing phase could be reliably rejected.

We have thus far performed the work in simulation using 2D, planar models to explore human locomotion optimization in the most important plane. Of course, simulations are usually not easily transferred to reality, and future work includes the transfer of our developed controllers to the CoMan robot to validate if our obtained gaits are suitable for walking using the robot. There are several difficulties in doing so. First, we do not optimize for walking in 3D and the resulting controller thus lacks control of the third dimension. Furthermore, even though our controllers are stable in simulation, and are shown to exhibit a certain robustness, we do not expect these local and open-loop controllers to be sufficient for stabilization of the real robot. A possible way forward would be to use our optimized impedance controllers as feed-forward pattern generators while a second, global stabilization feed-back controller modulates the control signals to obtain a desired global stability.

Finally, we come back to the co-design of the lower extremities, wearable robot for human locomotion assistance. We combined all of our developed tools, methods and algorithms in the design of a methodology for the effective optimization of various wearable robot structures and their control. We limited ourselves to providing support for the hip and knee human joints, leaving the ankle conventionally actuated. A candidate selection of possibly interesting topologies resulted from an exhaustive enumeration taking into consideration the range of motion and mobility of the structure. Furthermore, we developed a method to obtain all possible pairs of actuated joints which fully described the motion of the system. By using cȯdγn for the modeling, we could easily parametrize the morphology for each topology as well as the control. Furthermore, cȯdγn provided all of the methods necessary to verify singularities of the wearable robot structure *and* those of the human joints during nominal gait, leading to an important pruning step in viable morphological solutions. We then performed an intensive optimization using MMPSO to explore 126 morphological subspaces, each with their own set of parameters, finally obtaining a number of non-anthropomorphic wearable robot solutions.

Being a complex optimization problem, and due to the stochastic nature of the optimization method, we do not always obtain good solutions which are able to provide a stable walking gait. Of those that do, we observed that a reasonable human-like gait (given the complexity) can be obtained while still optimizing only for high-level objectives. The obtained wearable robot structures expose various interesting characteristics, from which the most prominent is seen in the mass distribution: each solution has optimized such as to minimize changing the center of mass location of the complete system (human and wearable robot), leading to increased stability. Furthermore, at least one actuator is always attached to the torso.

There are still many venues for exploration of the co-design approach for the possible design of a non-anthropomorphic wearable robot, such as exploration of interaction forces, energy stores and proper ankle support. Our optimized non-anthropomorphic solutions are not more energy efficient than an anthropomorphic design using the same materials. The methodology that we have developed is meant as an aid in the design process, where iterative refinements and optimizations can be used to explore a morphological and parametric space which is otherwise too vast.

A Models

cȯdyn model A.1 – Basic CoMan model definition in cȯdyn. This makes use of an additional file (provided below in model A.2) which specifies the inertial and kinematic properties. This file constructs the rigid body dynamics on top of the base model.

```
require "physics/physics.cdn"
require "physics/contacts.cdn"

# The model properties are defined in a separate file. This allows
# for a more modular reuse of the model parameters, for other
# purposes than forward dynamics.
include "model.cdn"

# Here we apply physics specific templates to the various nodes
# defined in the model.
node | "coman" : physics.system {
    # Apply joint templates to the model
    node | "torso"  : physics.joints.planarY {}
    node | "@limbs" : physics.joints.revoluteY {}

    # Kinematic structure
    edge from "torso"            to "hip{Right,Left}" : physics.joint {}
    edge from "hip{Right,Left}"  to "knee@1"          : physics.joint {}
    edge from "knee{Right,Left}" to "ankle@1"         : physics.joint {}

    # Add two hard contact points in each of the ankles. We use defines
    # from model.cdn for the dimensions of the foot so they are easily
    # configurable
    node | "ankle{Right,Left}" {
        # Contact point on the back of the heel
        node "c1" : physics.contacts.hardPlanarY {
            soleXMin = "-@heellength"
            location = "[soleXMin; 0; -@footheight]"
        }

        # Contact point at the front of the foot
        node "c2" : physics.contacts.hardPlanarY {
            soleXMax = "@footlength - @heellength - @footlength * @toelength"
            location = "[soleXMax; 0; -@footheight]"
        }
    }
```

```
        include "physics/model.cdn"
        include "physics/dynamics.cdn"
    }
```

cȯdγn model A.2 – CoMan inertial and kinematic properties. This file is used by model A.1 to construct the full rigid body dynamics model.

```
    defines {
        limbs = "{hip,knee,ankle,toe}{Right,Left}"
        footlength = "0.15"
        heellength = "0.03"
        footheight = "0.095"
        toelength = 0.25
    }

    node "coman" {
        node "torso" {
            com = "[ 0.00919962; -0.000250202;  0.212992]"
              I = "[ 0.319291,    0.000513593, -0.00413699;
                     0.000513593, 0.169801,    -0.000237841;
                    -0.00413699, -0.000237841,  0.179851]"

             m = "14.4847"
            tr = "[0; 0; 0.5268]"
        }

        node "hipRight" {
            com = "[0.000739959;  0.00194785;  -0.100464]"
              I = "[0.0271576,    3.52594e-05,  0.000132356;
                    3.52594e-05,  0.0265146,   -0.00144774;
                    0.000132356, -0.00144774,   0.00298617]"

             m = "3.61731"
            tr = "[0; -0.0726; 0]"
        }

        node "hipLeft" {
            com = "[ 0.000739959; -0.00194785;  -0.100464]"
              I = "[ 0.0271576,   -3.16989e-05,  0.000132356;
                    -3.16989e-05,  0.0265146,    0.00158918;
                    0.000132356,   0.00158918,   0.00298617]"

             m = "3.61731"
            tr = "[0; 0.0726; 0]"
        }

        node "kneeRight" {
            com = "[0.00246127;  -0.00530996;  -0.0859895]"
              I = "[0.0040604,    1.12272e-05,  1.2261e-05;
                    1.12272e-05,  0.00400565,   0.000518992;
                    1.2261e-05,   0.000518992,  0.00124141]"

             m = "1.40982"
            tr = "[0; 0; -0.2258]"
        }

        node "kneeLeft" {
            com = "[0.00246127;  0.00530996;  -0.0859895]"
              I = "[0.0040604,   1.12272e-05,  1.2261e-05;
```

```
                1.12272e-05, 0.00400565,   0.000518992;
                1.2261e-05,  0.000518992,  0.00124141]"

         m = "1.40982"
        tr = "[0; 0; -0.2258]"
    }

    node "ankleRight" {
        com = "[0.00508618; -0.00479956;  -0.0204323]"
          I = "[0.00229076,  3.25721e-05,  0.000333951;
                3.25721e-05, 0.00263102,   0.000153469;
                0.000333951, 0.000153469,  0.00174337]"

         m = "1.39639"
        tr = "[0; 0; -0.201]"
    }

    node "ankleLeft" {
        com = "[ 0.00508618;   0.00479956;  -0.0204323]"
          I = "[ 0.00229076,  -1.73029e-05,  0.000333951;
                -1.73029e-05,  0.00263102,   0.000398517;
                 0.000333951,  0.000398517,  0.00174337]"

         m = "1.39639"
        tr = "[0; 0; -0.201]"
    }
}
```

B Complementary figures

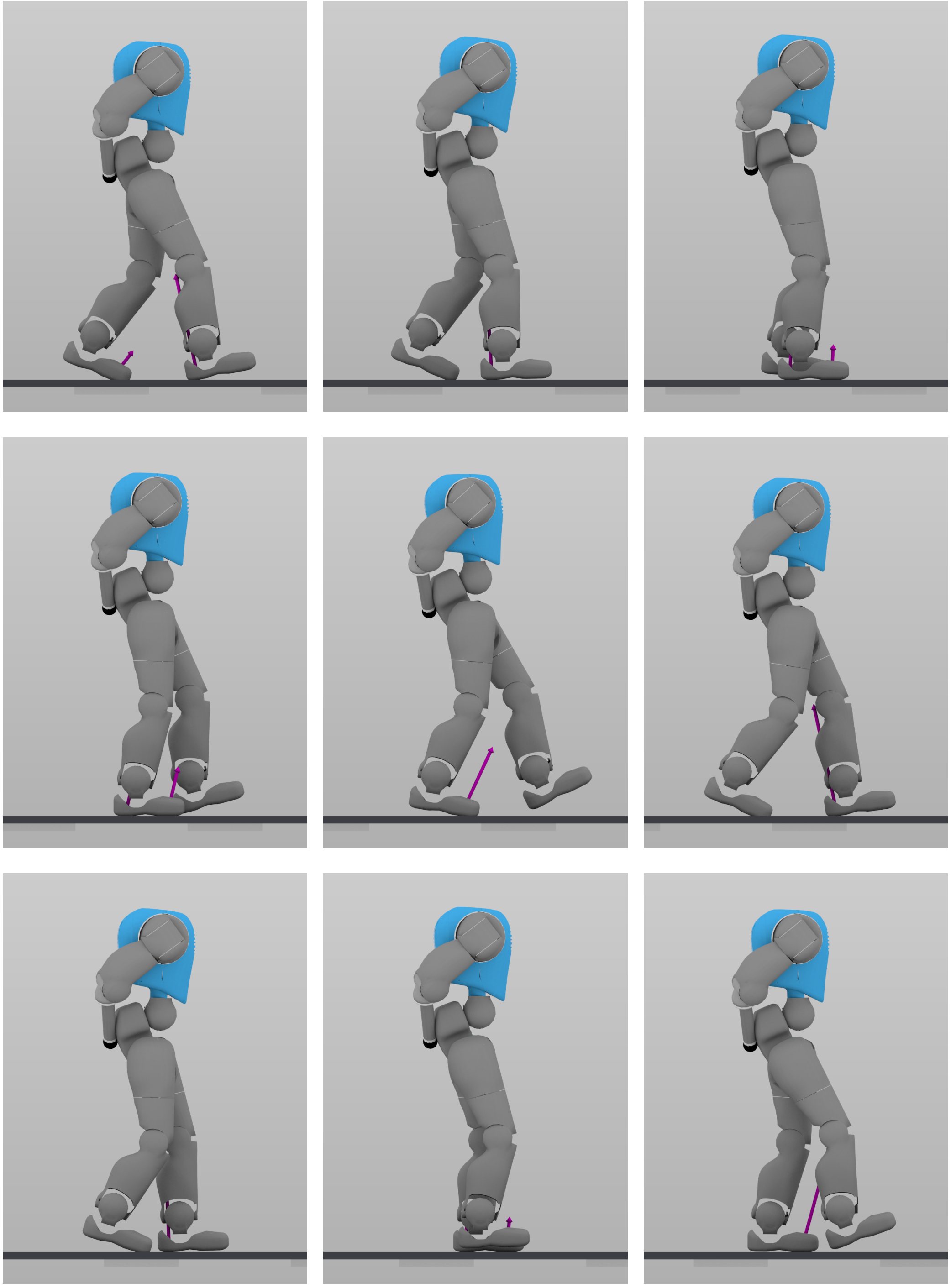


Figure B.1 – Rendering of a walking sequence optimized for the CoMan robot using a variable stiffness and damping controller (*step*). The gait shown here is relatively slow, $0.44\,\mathrm{m\,s^{-1}}$, but manages an efficient cost of transport. See also section 4.2 for more details on the methods used to optimize this gait.

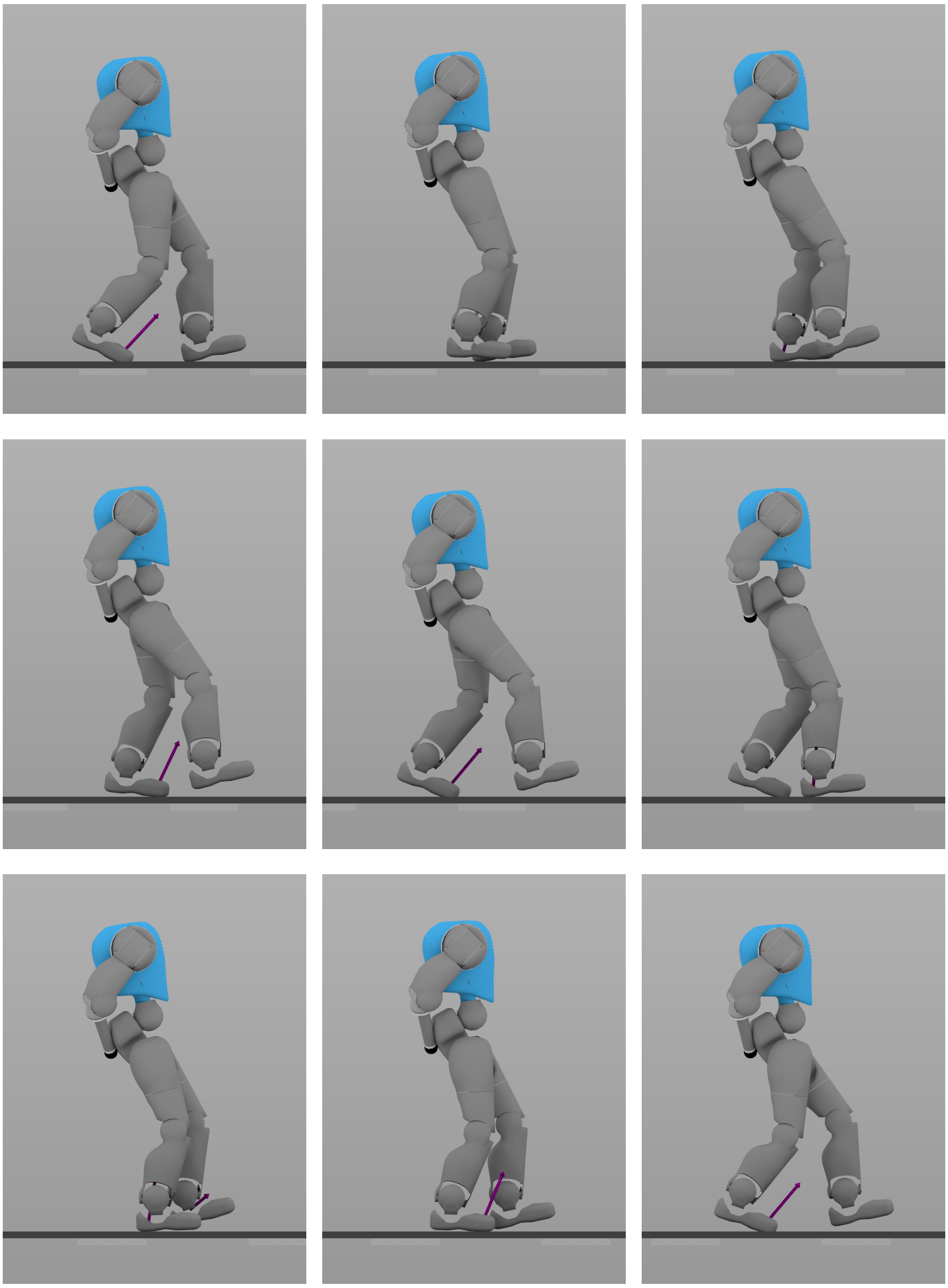


Figure B.2 – Rendering of a walking sequence optimized for the CoMan robot using a variable stiffness and damping controller (*step*) while under perturbation. The perturbations are applied randomly on the ankle in the direction of locomotion during the swing phase. The resulting controller, although open loop, is able to self-stabilize by modulating the stiffness and damping periodically. See also section 4.2.4 for more details on the methods used to optimize this gait.

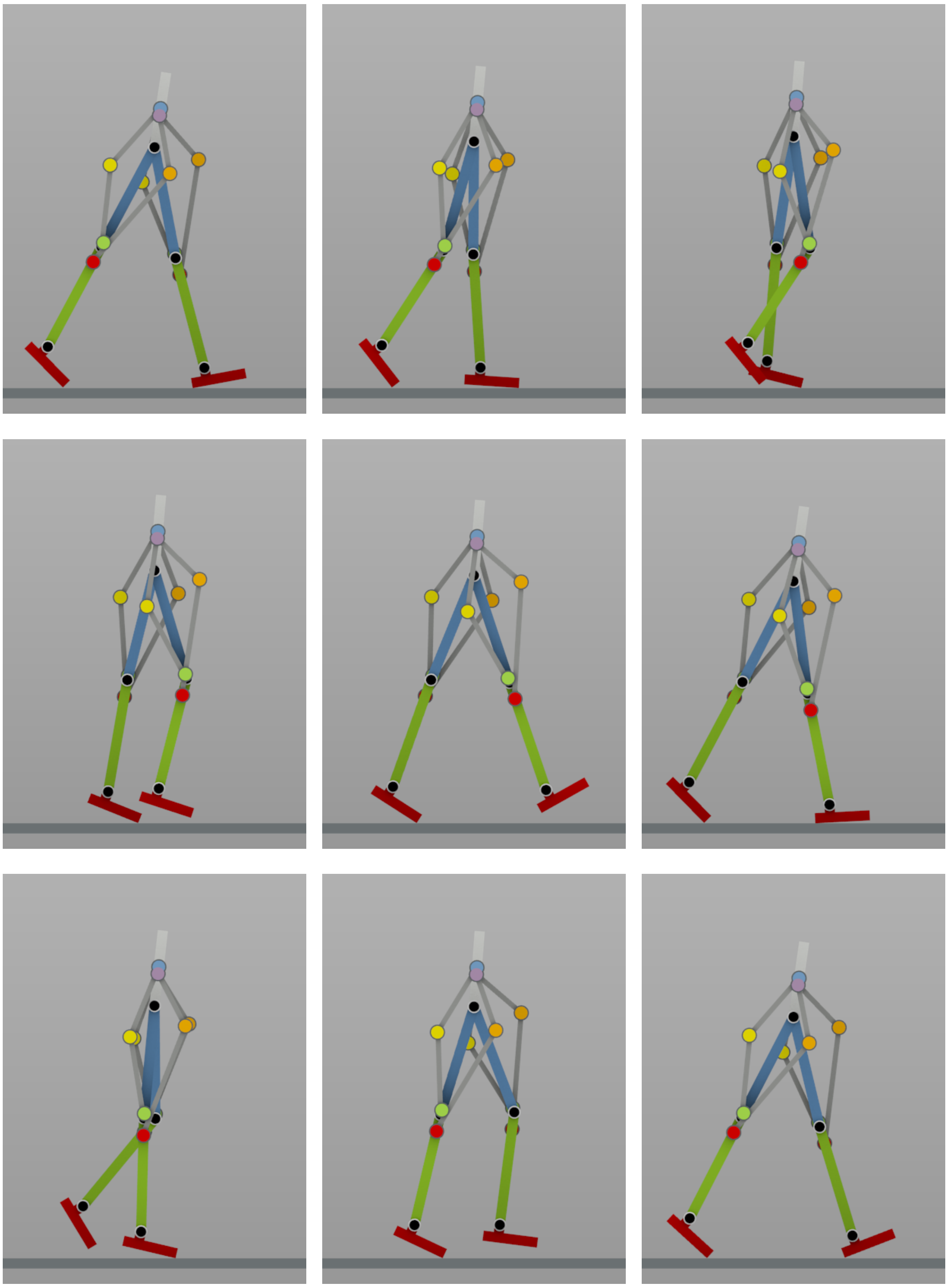


Figure B.3 – Rendering of a non-anthropomorphic wearable robot co-designed for human locomotion. See chapter 5 for more details on the methods used to optimize this wearable robot.

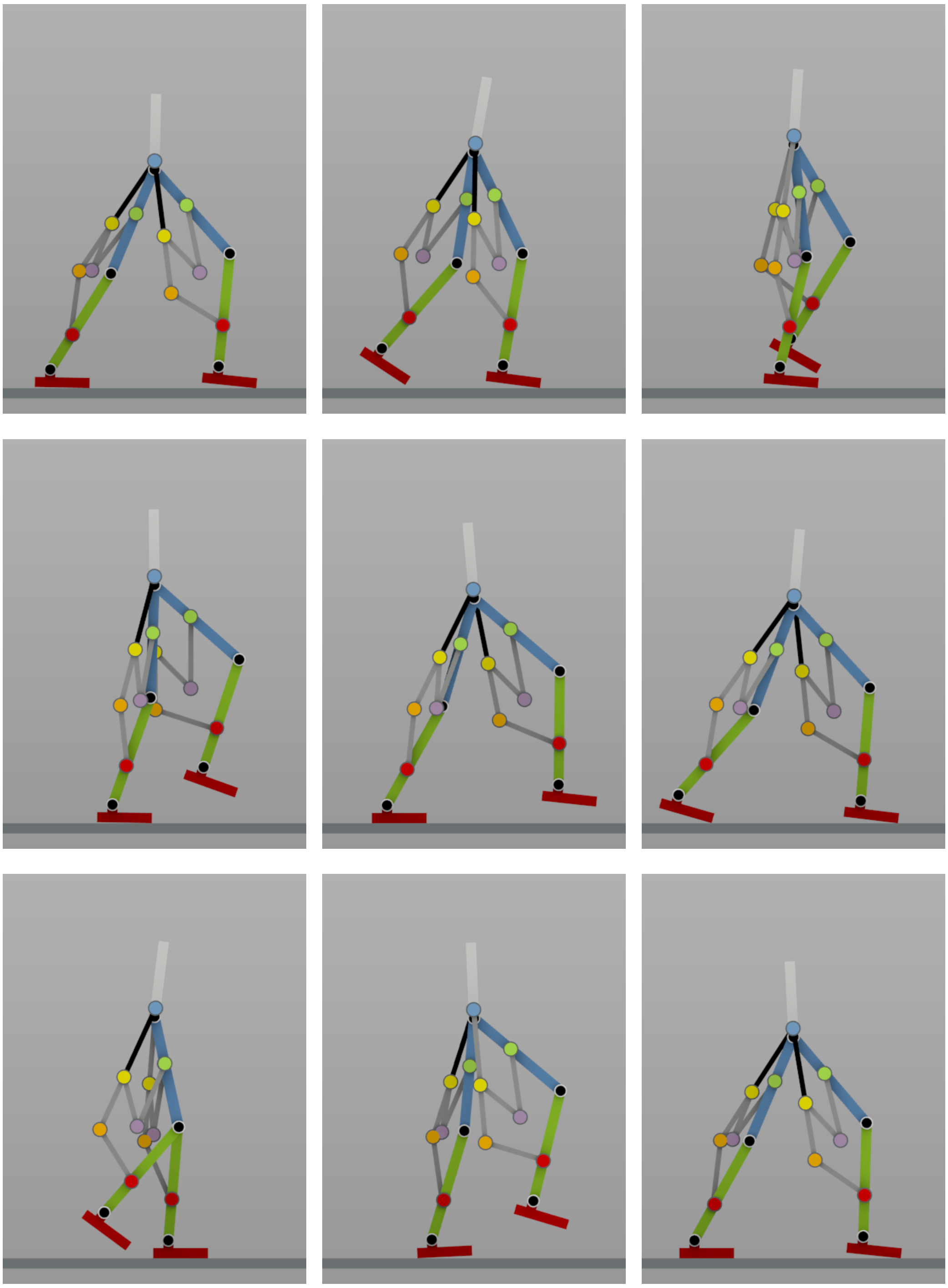


Figure B.4 – Rendering of a non-anthropomorphic wearable robot co-designed for human locomotion. See chapter 5 for more details on the methods used to optimize this wearable robot.

Bibliography

Mostafa Ajallooeian, Jesse van den Kieboom, Albert Mukovskiy, Martin A Giese, and Auke J Ijspeert. A general family of morphed nonlinear phase oscillators with arbitrary limit cycle shape. *Physica D: Nonlinear Phenomena*, 263:41–56, 2013.

Frank C Anderson and Marcus G Pandy. Dynamic optimization of human walking. *Journal of biomechanical engineering*, 123(5):381–390, 2001.

Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

Yvan Bourquin, Auke Jan Ijspeert, and Inman Harvey. Self-organization of locomotion in modular robots. *Unpublished Diploma Thesis, http://birg. epfl. ch/page53073. html*, 2004.

Rodney A Brooks. Artifical life and real robots. In *Toward a practice of autonomous systems: Proc. of the 1st Europ. Conf. on Artificial Life*, page 3, 1992.

Jonas Buchli, Ludovic Righetti, and Auke Jan Ijspeert. A dynamical systems approach to learning: a frequency-adaptive hopper robot. In *Advances in Artificial Life*, pages 210–220. Springer, 2005.

Jonas Buchli, Evangelos Theodorou, Freek Stulp, and Stefan Schaal. Variable impedance control-a reinforcement learning approach. In *Robotics: Science and Systems Conference (RSS)*. Citeseer, 2010.

John C Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2008.

GA Cavagna, P Franzetti, and T Fuchimoto. The mechanics of walking in children. *The Journal of physiology*, 343(1):323–339, 1983.

M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1): 58–73, 2002.

Maurice Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

Maurice Clerc. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering*, pages 219–239. Springer, 2004.

Carlos A Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems*, 1(3):269–308, 1999.

Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

Steven H Collins, Martijn Wisse, and Andy Ruina. A three-dimensional passive-dynamic walking robot with two legs and knees. *The International Journal of Robotics Research*, 20(7): 607–615, 2001.

Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *Biomedical Engineering, IEEE Transactions on*, 54(11):1940–1950, 2007.

JM Donelan, Q Li, V Naing, JA Hoffer, DJ Weber, and AD Kuo. Biomechanical energy harvesting: generating electricity during walking with minimal user effort. *Science*, 319(5864):807–810, 2008.

Marco Dorigo and Mauro Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. Springer, 2010.

Vincent Duchaine and Clement M Gosselin. General model of human-robot cooperation using a novel velocity based variable impedance control. In *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint*, pages 446–451. IEEE, 2007.

R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, pages 84 – 88, 2000.

G.I. Evers and M. Ben Ghalia. Regrouping particle swarm optimization: A new global optimization algorithm with improved performance consistency across benchmarks. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 3901–3908, 2009.

Roy Featherstone. *Rigid body dynamics algorithms*, volume 49. Springer New York, 2008.

Dario Floreano, Francesco Mondada, Andres Perez-Uribe, and Daniel Roggen. Evolution of embodied intelligence. In *Embodied artificial intelligence*, pages 293–311. Springer, 2004.

Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.

H. Geyer and H. Herr. A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 18(3):263–273, 2010.

David Edward Goldberg *et al. Genetic algorithms in search, optimization, and machine learning*, volume 412. Addison-wesley Reading Menlo Park, 1989.

Erico Guizzo and Harry Goldstein. The rise of the body bots [robotic exoskeletons]. *Spectrum, IEEE*, 42(10):50–56, 2005.

Fumio Hara and Rolf Pfeifer. *Morpho-functional Machines: The New Species: Designing Embodied Intelligence*. Springer, 2003.

Hugh Herr. Variable-mechanical-impedance artificial legs, April 1 2004. US Patent 20,040,064,195.

Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. The development of honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321–1326. IEEE, 1998.

Masato Hirose and Kenichi Ogawa. Honda humanoid robots development. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365 (1850):11–19, 2007.

Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4146–4151. IEEE, 2001.

Xiaohui Hu and Russell Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Computational Intelligence, Proceedings of the World on Congress on*, volume 2, pages 1677–1681. Ieee, 2002.

Qiang Huang, Kazuhito Yokoi, Shuuji Kajita, Kenji Kaneko, Hirohiko Arai, Noriho Koyachi, and Kazuo Tanie. Planning walking patterns for a biped robot. *Robotics and Automation, IEEE Transactions on*, 17(3):280–289, 2001.

Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

Ryojun Ikeura, Tomoki Moriguchi, and Kazuki Mizutani. Optimal variable impedance control for a robot and its application to lifting an object with a human. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 500–505. IEEE, 2002.

Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620–1626. IEEE, 2003.

Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

H Kazerooni and R Steger. The berkeley lower extremity exoskeleton. *Journal of dynamic systems, measurement, and control*, 128(1):14–25, 2006.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *"Proceedings of IEEE International Conference on Neural Networks"*, volume 4, pages 1942–1948, 1995.

J. Kennedy and RC Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference On Systems Man And Cybernetics*, volume 5, 1997.

Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of*, 3(1):43–53, 1987.

Jérémie Knüsel. *Modeling a diversity of salamander motor behaviors with coupled abstract oscillators and a robot*. PhD thesis, STI, Lausanne, 2013.

John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

Karl Kutzbach. Mechanische leitungsverzweigung, ihre gesetze und anwendungen. *Maschinenbau. Betrieb*, 8:710–716, 1929.

Ken Larpin, Soha Pouya, Jesse van den Kieboom, and Auke Jan Ijspeert. Co-evolution of morphology and control of virtual legged robots for a steering task. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2799–2804. IEEE, 2011.

NM Abdul Latiff, CC Tsimenidis, and BS Sharif. Performance comparison of optimization algorithms for clustering in wireless sensor networks. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–4. IEEE, 2007.

W-F Leong and Gary G Yen. Impact of tuning parameters on dynamic swarms in PSO-based multiobjective optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1317–1324. IEEE, 2008.

A Lindenmayer. Mathematical models for cellular interactions in development. i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, mar 1968. PMID: 5659071.

Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, aug 2000.

Jason D Lohn, Gregory S Hornby, and Derek S Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In *Genetic Programming Theory and Practice II*, pages 301–315. Springer, 2005.

R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

Ernesto C Martinez-Villalpando and Hugh Herr. Agonist-antagonist active knee prosthesis: A preliminary study in level-ground walking. *J Rehabil Res Dev*, 46(3):361–73, 2009.

Tad McGeer. Passive dynamic walking. *the international journal of robotics research*, 9(2): 62–82, 1990.

O. Michel. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.

Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer, 1999.

Rico Moeckel, Yura N Perov, Anh The Nguyen, Massimo Vespignani, Stephane Bonardi, Soha Pouya, Alexander Sproewitz, Jesse van den Kieboom, Frederic Wilhelm, and Auke Jan Ijspeert. Gait optimization for roombots modular robots—matching simulation and reality. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3265–3272. IEEE, 2013.

Christopher K. Monson and Kevin D. Seppi. Adaptive diversity in PSO. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 59–66, Seattle, Washington, USA, 2006. ACM.

Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. DNPSO: a dynamic niching particle swarm optimizer for multi-modal optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 26–32. IEEE, 2008.

Lee Nolan. Carbon fibre prostheses and running in amputees: a review. *Foot and ankle surgery*, 14(3):125–129, 2008.

Chao Ou and Weixing Lin. Comparison between PSO and GA for parameters optimization of PID controller. In *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, pages 2471–2475. IEEE, 2006.

Jong H Park and Kyoung D Kim. Biped robot walking using gravity-compensated inverted pendulum mode and computed torque control. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3528–3533. IEEE, 1998.

Jong Hyeon Park. Impedance control for biped robot locomotion. *Robotics and Automation, IEEE Transactions on*, 17(6):870–882, 2001.

Konstantinos E Parsopoulos and Michael N Vrahatis. Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607. ACM, 2002.

Chandana Paul and Josh C Bongard. The road less travelled: Morphology in the optimization of biped robot locomotion. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 226–232. IEEE, 2001.

Rolf Pfeifer, Max Lungarella, and Fumiya Iida. Self-organization, embodiment, and biologically inspired robotics. *science*, 318(5853):1088–1093, 2007.

Soha Pouya, Jesse van den Kieboom, A Spröwitz, and Auke Jan Ijspeert. Automatic gait generation in modular robots:“to oscillate or to rotate; that is the question”. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 514–520. IEEE, 2010.

Jim Pugh and Alcherio Martinoli. Discrete multi-valued particle swarm optimization. In *Proceedings of IEEE swarm intelligence symposium*, volume 1, pages 103–110, 2006.

Md Mozasser Rahman, Ryojun Ikeura, and Kazuki Mizutani. Investigation of the impedance characteristic of human arm for development of robots to cooperate with humans. *JSME International Journal Series C*, 45(2):510–518, 2002.

Tapabrata Ray and K. M. Liew. A swarm metaphor for multiobjective design optimization. *Engineering Optimization*, 34(2):141, 2002.

Margarita Reyes-Sierra and CA Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

L. Righetti and A.J. Ijspeert. Programmable central pattern generators: an application to biped locomotion control. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, 2006.

R. Ronsse, N. Vitiello, T. Lenzi, J. van den Kieboom, M.C. Carrozza, and A.J. Ijspeert. Adaptive oscillators with human-in-the-loop: Proof of concept for assistance and rehabilitation. In *Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on*, pages 668–674, Sept 2010.

R. Ronsse, B. Koopman, N. Vitiello, T. Lenzi, S. M M De Rossi, J. van den Kieboom, E. van Asseldonk, M.C. Carrozza, H. van der Kooij, and A.J. Ijspeert. Oscillator-based walking assistance: A model-free approach. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–6, June 2011a.

R. Ronsse, T. Lenzi, N. Vitiello, B. Koopman, E. Van Asseldonk, S.M.M. De Rossi, J. Van Den Kieboom, H. Van Der Kooij, M.C. Carrozza, and A.J. Ijspeert. Oscillator-based assistance of cyclical movements: model-based and model-free approaches. *Medical and Biological Engineering and Computing*, pages 1–13, 2011b.

R. Ronsse, N. Vitiello, T. Lenzi, J. van den Kieboom, M.C. Carrozza, and A.J. Ijspeert. Human - robot synchrony: Flexible assistance using adaptive oscillators. *Biomedical Engineering, IEEE Transactions on*, 58(4):1001–1012, April 2011c.

Justinian P Rosca and Dana H Ballard. *Discovery of subroutines in genetic programming*, chapter 9. The MIT Press, Cambridge, Massachusetts, 1996.

Jean-Claude Samin. *Symbolic modeling of multibody systems*, volume 112. Springer, 2003.

Andre Schiele and Frans CT van der Helm. Kinematic design to improve ergonomics in human machine interaction. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 14(4):456–469, 2006.

Knut Schmidt-Nielsen. Locomotion: energy cost of swimming, flying, and running. *Science*, 177(4045):222–228, 1972.

Fabrizio Sergi, Dino Accoto, Nevio L Tagliamonte, Giorgio Carpino, and Eugenio Guglielmelli. A systematic graph-based method for the kinematic synthesis of non-anthropomorphic wearable robots for the lower limbs. *Frontiers of Mechanical Engineering*, 6(1):61–70, 2011.

A. Seyfarth, H. Geyer, M. Günther, and R. Blickhan. A movement criterion for running. *Journal of Biomechanics*, 35(5):649–655, 2002.

Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *The 1998 IEEE International Conference on Evolutionary Computation, ICEC'98*, pages 69–73, 1998.

K. Sims. Evolving 3d morphology and behavior by competition. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 28 – 39, 1994a.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994b.

Lee Spector, Howard Barnum, Herbert J Bernstein, Nikhil Swamy, *et al.* Finding a better-than-classical quantum and/or algorithm using genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2239–2246, 1999.

Alexander Sproewitz, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert. Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4259–4264. IEEE, 2009.

Alexander Spröwitz, Soha Pouya, Stéphane Bonardi, Jesse Van den Kieboom, Rico Möckel, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert. Roombots: reconfigurable robots for adaptive furniture. *Computational Intelligence Magazine, IEEE*, 5(3):20–32, 2010.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

Nikos G Tsagarakis, Stephen Morfey, Gustavo Medrano Cerda, Li Zhibin, and Darwin G Caldwell. Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 673–678. IEEE, 2013.

F. van den Bergh and A.P. Engelbrecht. A cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):225–239, 2004.

Antonie J van den Bogert. Exotendons for assistance of human locomotion. *BioMedical Engineering OnLine*, 2:17, 2003. PMC270067.

Jesse van den Kieboom and Auke Jan Ijspeert. Exploiting Natural Dynamics in Biped Locomotion using Variable Impedance Control. In *IEEE Conference on Humanoids Robots*, 2013.

Jesse van den Kieboom, Soha Pouya, and Auke Jan Ijspeert. Meta morphic particle swarm optimization. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, pages 231–244. Springer, 2013.

Miomir Vukobratović and Branislav Borovac. Zero-moment point—thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01):157–173, 2004.

Miomir Vukobratovic and Davor Juricic. Contribution to the synthesis of biped gait. *Biomedical Engineering, IEEE Transactions on*, (1):1–6, 1969.

Conor James Walsh, Kenneth Pasch, and H Herr. An autonomous, underactuated exoskeleton for load-carrying augmentation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1410–1415. IEEE, 2006.

Conor James Walsh, Ken Endo, and Hugh Herr. A quasi-passive leg exoskeleton for load-carrying augmentation. *International Journal of Humanoid Robotics*, 4(03):487–506, 2007.

Jack M Wang, Samuel R Hamner, Scott L Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics (TOG)*, 31(4):25, 2012.

David A Winter. *Biomechanics and motor control of human movement*. John Wiley & Sons, 2009.

M Wisse, AL Schwab, and RQ vd Linde. A 3d passive dynamic biped with yaw and roll compensation. *Robotica*, 19(3):275–284, 2001.

Martijn Wisse. Essentials of dynamic walking; analysis and design of two-legged robots. 2004.

Martijn Wisse. Three additions to passive dynamic walking: actuation, an upper body, and 3d stability. *International Journal of Humanoid Robotics*, 2(04):459–478, 2005.

Chukiat Worasucheep. A particle swarm optimization with stagnation detection and dispersion. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 424–429. IEEE, 2008.

Kemin Zhou, John Comstock Doyle, Keith Glover, *et al.* *Robust and optimal control*, volume 272. Prentice Hall New Jersey, 1996.

Curriculum Vitae

Jesse van den Kieboom

Born 26 June 1984, Breda, The Netherlands

Email: jessevdk@gmail.com

Experience

Swiss Federal Institute of Technology (EPFL), Lausanne, Suisse 2009 - 2014
Biorobotics Laboratory

Phd thesis on the dynamics of human locomotion and co-design of lower limb assistive devices.

Rijksuniversiteit Groningen (RUG), Groningen, The Netherlands 2005

Member of the winning team of the Dutch RoboChallenge 2005, a mobile robot competition with a self-designed, manufactured and controlled robotic platform.

Education

Swiss Federal Institute of Technology (EPFL), Lausanne, Suisse

PhD student in the Biorobotics laboratory	2009 - 2014
Master thesis at the Biorobotics laboratory *Biped Locomotion and Stabilization - A Practical Approach*	2008 - 2009

Rijksuniversiteit Groningen (RUG), Groningen, The Netherlands

Bachelor in Artificial Intelligence	2003 - 2006
Master of science in Artificial Intelligence	2007 - 2009

Journal publications

Ronsse, R. & Lenzi, T. & Vitiello, N. & Koopman, B. & Van Van Asseldonk, E. & Rossi, S.M.M. & Den Kieboom, J. & Van Kooij, H. & Carrozza, M.C. & Ijspeert, A.J. (2011). Oscillator-based assistance of cyclical movements: model-based and model-free approaches. *Medical and Biological Engineering and Computing*, 1—13.

Ajallooeian, M. & van Kieboom, J. & Mukovskiy, A. & Giese, M. A. & Ijspeert, A. J. (2013). A general family of morphed nonlinear phase oscillators with arbitrary limit cycle shape. *Physica D: Nonlinear Phenomena*,*263*, 41—56.

Conference publications

Pouya, S. & van Kieboom, J. & Sprowitz, A. & Ijspeert, A. J. (2010). Automatic gait generation in modular robots:“to oscillate or to rotate; that is the question”. *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 514—520.

Ronsse, R. & Vitiello, N. & Lenzi, T. & van Kieboom, J. & Carrozza, M.C. & Ijspeert, A.J. (2010). Adaptive oscillators with human-in-the-loop: Proof of concept for assistance and rehabilitation. *Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on*, 668-674.

Larpin, K. & Pouya, S. & van Kieboom, J. & Ijspeert, A. J. (2011). Co-evolution of morphology and control of virtual legged robots for a steering task. *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, 2799—2804.

Ronsse, R. & Koopman, B. & Vitiello, N. & Lenzi, T. & De De Rossi, S. M M & van Kieboom, J. & Asseldonk, E. & Carrozza, M.C. & van Kooij, H. & Ijspeert, A.J. (2011). Oscillator-based walking assistance: A model-free approach. *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, 1-6.

Ronsse, R. & Vitiello, N. & Lenzi, T. & van Kieboom, J. & Carrozza, M.C. & Ijspeert, A.J. (2011). Human - Robot Synchrony: Flexible Assistance Using Adaptive Oscillators. *Biomedical Engineering, IEEE Transactions on*,*58*,1001-1012.

van Kieboom, J. & Pouya, S. & Ijspeert, A. J. (2013). Meta Morphic Particle Swarm Optimization. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, 231—244.

Gay, S. & van Kieboom, J. & Santos-Victor, J. & Ijspeert, A. (2013). Model-Based and Model-Free Approaches for Postural Control of a Compliant Humanoid Robot using Optical Flow. *IEEE Conference on Humanoids Robots*.

van Kieboom, J. & Ijspeert, A. J. (2013). Exploiting Natural Dynamics in Biped Locomotion using Variable Impedance Control. *IEEE Conference on Humanoids Robots*.

Moeckel, R. & Perov, Y. N. & Nguyen, A. T. & Vespignani, M. & Bonardi, S. & Pouya, S. & Sproewitz, A. & van Kieboom, J. & Wilhelm, F. & Ijspeert, A. J. (2013). Gait optimization for roombots modular robots—Matching simulation and reality. *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 3265—3272.

Magazine publications

Spröwitz, A. & Pouya, S. & Bonardi, S. & van Kieboom, J. & Möckel, R. & Billard, A. & Dillenbourg, P. & Ijspeert, A. J. (2010). Roombots: reconfigurable robots for adaptive furniture. *Computational Intelligence Magazine, IEEE*,*5*, 20—32.