Improved Humanoid Robot Movement through Impact Perception and Walk Optimisation

Jason Kulk
B Eng (Computer) and B Sci (Physics)

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
School of Electrical Engineering and Computer Science
University of Newcastle, Australia
May 2012

The thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis, when deposited in the University Library[a], being made available for loan and photocopying subject to the provisions of the Copyright Act 1968.

[a]Unless an Embargo has been approved for a determined period

Signature: _____

Jason Kulk

Contents

1 Introduction **1**

1.1 RoboCup Soccer: Synthetic Application for Humanoid Robots . 3

1.1.1 NUbots: University of Newcastle's RoboCup Team . . . 4

1.2 Software for Legged Robots 5

1.3 Stance for Humanoid Robots 7

1.4 Walk Optimisation for Humanoid Robots 9

1.5 Summary of Contributions and Publications 13

1.5.1 Publications 13

1.6 Thesis Overview 15

2 Review of Human Motion **17**

2.1 Introduction 18

2.2 Human Quiet Stance 19

2.2.1 Joint Positions for Stance 19

2.2.2 Proprioception for Detecting Perturbations 19

2.2.3 Joint Stiffness for Stance 21

2.3 Human Perturbed Stance 22

2.3.1 Responses to Perturbations 22

2.4 Human Walking 23

2.4.1 An Efficiency–based Fitness Function 23

2.4.2 Joint Stiffness for Walking 24

2.5 Summary 24

3 The NUPlatform Software Framework **25**

3.1 Introduction 26

3.1.1 Related Work 27

3.1.2 Architecture Overview 29
3.2 The Blackboard 30
3.2.1 Sensors 31
3.2.2 Actuators 32
3.2.3 Visual Information 34
3.2.4 Jobs 35
3.2.5 Network Information 35
3.3 The Platform 36
3.3.1 NUPlatform 37
3.3.2 NUCamera 38
3.3.3 NUSensors 38
3.3.4 NUActionators 39
3.4 The Software Modules 39
3.4.1 Behaviour 39
3.4.2 Motion 40
3.5 System Configuration 42
3.6 Applications of NUPlatform 42
3.7 Conclusion 44

4 Impact Perception for a Standing Humanoid Robot 45
4.1 Introduction 46
4.1.1 Review of Related Work 46
4.1.2 System Overview 47
4.2 Equipment and Data Collection 48
4.3 Detecting a Perturbation 53
4.3.1 An Optimised Threshold Detector 53
4.3.2 Discussion of Detection Results 55
4.4 Perceiving the Location of a Perturbation 58
4.4.1 Classification of Location Using an SVM 58
4.4.2 Discussion of Classification Results 60
4.5 Estimating the Direction and Strength of a Perturbation 63
4.5.1 Estimation of a Perturbation Using SVR Models 63
4.5.2 Discussion of Estimation Results 64
4.6 Effect of Low Joint Stiffness on Performance 66
4.7 Conclusion 67

5 Improvements in Walking through Joint Stiffness Reduction 69
5.1 Introduction . . . 70
5.2 Equipment and Method . . . 71
5.2.1 Hardware and Software . . . 71
5.2.2 Optimisation Algorithm and Parameter Space . . . 72
5.2.3 Optimisation Path and Fitness Function . . . 73
5.3 Results . . . 73
5.3.1 Speed . . . 74
5.3.2 Efficiency . . . 74
5.3.3 Stability . . . 76
5.4 Discussion . . . 80
5.5 Conclusion . . . 81

6 Meta–optimisation of Walk Optimisation Techniques 83
6.1 Introduction . . . 84
6.2 Equipment and Method . . . 85
6.2.1 Hardware and Software . . . 85
6.2.2 Optimisation Path . . . 86
6.2.3 Optimisation Expense . . . 88
6.3 Optimisation Algorithms . . . 89
6.3.1 Evolutionary Hill Climbing with Line Search . . . 90
6.3.2 Policy Gradient Reinforcement Learning . . . 90
6.3.3 Gaussian Particle Swarm Optimisation . . . 93
6.4 Fitness Functions for Optimisation . . . 93
6.4.1 Speed . . . 94
6.4.2 Efficiency . . . 94
6.4.3 Froude–Number . . . 95
6.5 Parameter Spaces for Optimisation . . . 95
6.6 Meta-optimisation of Algorithms . . . 96
6.7 Design of the Comparison of Walk Optimisation Techniques . . . 97
6.8 Comparison of Algorithms . . . 99
6.9 Comparison of Fitness Functions . . . 108
6.10 Comparison of Parameter Spaces . . . 110
6.11 Application to the Physical NAO . . . 112
6.12 Conclusion . . . 115

7 Walk Optimisation with Redundant Fitness Functions 117
7.1 Introduction . . . 118
7.2 Equipment and Method . . . 119
7.3 Opposition–based PGRL with Redundant Fitness Functions . . 121
7.3.1 Opposition–based Policy Generation . . . 121
7.3.2 Use of Redundant Fitness . . . 121
7.4 Applications of the Improved PGRL Algorithm . . . 124
7.5 Conclusion . . . 129

8 Gait–Phase Dependent Joint Stiffnesses 130
8.1 Introduction . . . 131
8.1.1 Phases of the Gait Cycle . . . 132
8.2 Phase Dependent Stiffness with Fixed Traditional Walk Parameters . . . 134
8.2.1 Equipment and Method . . . 134
8.2.2 Results . . . 140
8.3 Optimisation of Phase Dependent Stiffness and Traditional Walk Parameters . . . 140
8.3.1 Equipment and Method . . . 140
8.3.2 Results . . . 142
8.4 Discussion . . . 144
8.4.1 Phase Dependent Stiffness with Fixed Traditional Walk Parameters . . . 144
8.4.2 Phase Dependent Stiffness with Variable Traditional Walk Parameters . . . 145
8.4.3 Robot Tracking . . . 146
8.5 Conclusion . . . 147

9 Conclusion 148
9.1 Conclusions . . . 148
9.2 Future Work . . . 151
9.3 Summary . . . 152

Bibliography 153

Abstract

The proficiency of humanoid robot movement, which is currently quite elementary, needs to be improved if humanoid robots are to fulfil most of their intended applications. Two of the more essential motor skills of a humanoid robot are related to its ability to stand and walk. Enhancement of these abilities is the focus of the work presented in this thesis.

We first investigate the use of the proprioceptive sense, in particular the joint velocities, to perceive and quantify external perturbations to a standing humanoid robot. A system consisting of an optimised threshold detector, a Support Vector Machine and a pair of orthogonal Support Vector Regression models is developed to utilise this proprioceptive sense. We demonstrate, through the implementation on a physical robot, that the proposed system is able to detect, locate and estimate the magnitude and direction of any given impact.

Next we consider improvements to humanoid robot walking through the enhancement of walk optimisation techniques. To this end, in simulation, a meta–optimisation is performed to determine: an appropriate set of tuning parameters for three different optimisation algorithms, the most suitable optimisation algorithm, a relevant fitness function and a pertinent parameter space. The optimisation algorithms we consider include: Evolutionary Hill Climb with Line Search, Particle Swarm Optimisation and Policy Gradient Reinforcement Learning (PGRL). We evaluated fitness functions based on the walk speed, efficiency and Froude–number. The parameter space for the walk engine was assessed with and without additional joint stiffness parameters. We found that the best walk optimisation technique consisted of PGRL with an efficiency based fitness function utilising additional joint stiffness parameters.

We achieved further improvements on the walk optimisation by applying the safe redundancy concept to extend PGRL. PGRL is a local optimisation algorithm, whereby incorporating safe redundancy allows the algorithm to escape from local extrema. We also expanded the parameter space to include gait–phase dependent joint stiffnesses. Furthermore, to facilitate a trade–off between the optimisation and the stress placed on the physical hardware, a measure of the wear experienced by the robot during the optimisation was introduced.

To verify the generality of the systems developed for the walk optimisation, they are evaluated on several different humanoid robot platforms: a simulated NAO, a physical NAO and a DARWIN-OP. The effectiveness of the proposed systems are demonstrated through their implementation in physical humanoid robot hardware and application to the RoboCup soccer competitions.

Chapter 1

Introduction

Contents

1.1 RoboCup Soccer: Synthetic Application for Humanoid Robots 3

1.1.1 NUbots: University of Newcastle's RoboCup Team . 4

1.2 Software for Legged Robots 5

1.3 Stance for Humanoid Robots 7

1.4 Walk Optimisation for Humanoid Robots 9

1.5 Summary of Contributions and Publications 13

1.5.1 Publications 13

1.6 Thesis Overview 15

Science fiction often describes a future where humans and intelligent robots live in collaboration to create a utopian–esque society. To thrive in a world built for humans, the robots are humanoids. That is, they have an anthropomorphic shape and walk on two legs. The robot's humanoid form allows it to efficiently manipulate man–made objects. Furthermore, the familiarity of the robot's humanoid appearance improves the ability of humans to interact with the robot [1, 2]. Consequently, humanoid robots have the potential to perform many tasks and have almost limitless applications.

There are numerous humanoid robot projects underway [3], closing the gap between fiction and reality. Notable examples include Honda's ASIMO [4] and AIST's HRP-4 [5], both of which have human–like appearances and are able to perform several simple tasks reasonably well.

There are many tangible applications for humanoid robots emerging in the literature. Broadly, the applications can be categorised as either acting in an assistive role, or as a replacement in a harsh environment, for humans. The former application has the humanoid robot working closely with humans, while the latter has the robot working more closely with the environment.

Possibly the largest potential application for humanoid robots is in healthcare. The humanoid robot could assume the role of a nurse or orderly [6], especially for in–home care of elderly patients. It could also play a more specialised role, for example, in the treatment of autism [7] or in the entertainment of young patients [8]. A similar application for a humanoid robot is as a personal assistant where the robot would play the role of a maid, nanny or secretary [9, 10].

The other primary area of application for humanoid robots is in industry and harsh environments. A humanoid robot has the potential to operate any machine designed for humans, for example, the operation of a fork–lift [11]. Consequently, a humanoid robot could be used to replace humans operating machines in dangerous environments, such as in mining or construction sites. Furthermore, a humanoid robot could be used directly in harsh environments, such as during a fire [12] and in space [13].

It is apparent that there are many potential applications for a capable humanoid robot. However, to fulfil these applications the robot needs to be able to stand and walk with a high level of proficiency. Presently, the state–of–the–art of humanoid movement is not adequate to meet the requirements

of most of the potential applications. This thesis addresses some of the issues relating to improved humanoid robot movement.

1.1 RoboCup Soccer: Synthetic Application for Humanoid Robots

RoboCup Soccer is an annual event where robots from across the world compete against each other to win a robotic soccer tournament. The RoboCup Soccer competition was created to inspire research in the areas of artificial intelligence and robotics. The ultimate goal of the competition is to have a robot team capable of winning against a world champion human team by the mid–21st century [14].

The motivation behind the creation of the RoboCup Soccer competition was two fold: to serve as a landmark project and to define a new standard problem [15]. A landmark project is one whose completion has little direct impact, however, the technological achievement of completing the project would be a landmark in its field. The accomplishment of the ultimate goal of the RoboCup Soccer competition would be a significant milestone in the field of robotics. Furthermore, the technologies developed to achieve the ultimate goal will have direct benefits through spinoff applications. Kitano et al. [15] saw the Apollo missions as the epitome of landmark projects, where the technologies developed to put a man on the moon have found applications in all areas of modern society [16].

The RoboCup soccer competitions have been running since 1997 resulting in many scientific publications [17, 18] being produced. There are also several publications where algorithms and techniques developed to succeed at RoboCup have been applied to other research domains [19], for example in robot design [20], machine learning [21], and localisation [22]. Thus, demonstrating that spin–off applications for the technology developed in the pursuit of the landmark are already being found.

The RoboCup Soccer competition is also an example of a standard problem. The RoboCup Soccer environment and criteria for success, are specified by a strict set of rules, for example see [23, 24]. Kitano et al. [15] suggested

Figure 1.1: RoboCup 2008 SPL.

that RoboCup Soccer be a replacement for computer chess as a standard artificial intelligence problem. As compared to computer chess, robot soccer is a multi–agent system in a noisy real–world environment, requiring both artificial intelligence and advanced physical robots.

The primary advantage of using a standard problem is that it allows for efficient and effective comparison of algorithms. RoboCup Soccer has been used as a standard problem for the software and walk optimisation techniques presented in this thesis. Consequently, it facilitates the direct comparison to the research of other teams at the RoboCup event itself, and the indirect comparison through related publications attempting to solve the same standard problem. Furthermore, the RoboCup Soccer competition is an external event, thus the effectiveness of the work presented in this thesis can be demonstrated outside the laboratory.

1.1.1 NUbots: University of Newcastle's RoboCup Team

The University of Newcastle has a long history in RoboCup soccer. In particular, competing as the NUbots in the Standard Platform League (SPL) from 2002 to 2011 and in the Kid–size Humanoid League in 2012. In 2008–2011 the NAO humanoid robot [25] was used as the standard platform for the SPL, as

pictured in Figure 1.1. In 2012 a team of DARWIN-OP robots [26] were used to enter the Kid–size Humanoid League.

The SPL is an interesting league because every team must use the same robot. This reduces the standard problem to only include the artificial intelligence component of the original RoboCup Soccer problem. Further strengthening the comparison of developed techniques and algorithms.

The University's involvement in these two leagues has provided access to two humanoid robots: the NAO and the DARWIN-OP. It is natural that these two robots feature prominently as test–beds in this thesis. Furthermore, RoboCup Soccer has been a synthetic application for the research performed in this thesis, representing a readily available real–world–like application.

1.2 Software for Legged Robots

The brains of a robot is the software running on its processors. The software takes sensory data and combines it with an internal state to generate output actions, typically implemented as a sense–think–act loop. The large number of sensors and actuators on a humanoid robot, and the complexity of the think step, dictate the use of a software framework [27]. The actual artificial intelligence of the robot is then built using the software framework.

A software framework is infrastructure that primarily organises the transfer of information. The framework handles the transfer of data between sensors and actuators hence it acts as an abstraction layer above the robotic hardware. The abstraction of underlying robotic hardware is essential for code portability and cross–robot support. The internal transfer of information between sub–modules and the operating system is also handled by the framework.

Opportunities for Cross–Robot Software

There is a common trend in the field of robotics to only use a single robot platform for the development, testing and verification of new algorithms and techniques. This may be due to the expense and availability of robot hardware. However, with more robots becoming commercially available at reduced costs, it has become possible to own several different robots. This provides potential

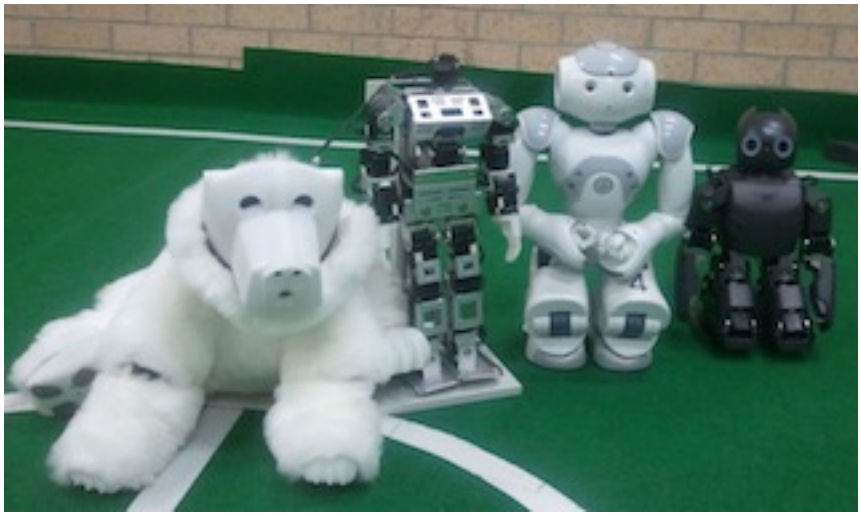


Figure 1.2: Robot platforms at the University of Newcastle. From left to right; the HyKim [28], a modified CycloidII [29], the NAO [30] and the DARWIN-OP [31].

for the implementation and verification of general techniques simultaneously on several robots.

The Robotics Laboratory at the University of Newcastle is a good example of this potential, where there are four similar robots available, as pictured in Figure 1.2. The ability to run the same software system on each platform would be of a great benefit.

The RoboCup Soccer competition provides motivation for the development of an enormous body of code designed to win a soccer match. However, the software written for RoboCup is typically not portable being tied to a specific robot platform. Recently, there has been work by several RoboCup teams to improve the portability of their software [32, 33], but the support for numerous physical robot platforms and operating systems remains a challenge.

The ability to use the RoboCup software in other research projects would both strengthen the quality of research in those projects and simplify the transition to spinoff applications that were behind the conception of RoboCup. These two benefits can be achieved through the development of a suitable software framework. A hardware abstraction layer can be used to isolate the intelligent algorithms from the robot hardware and a carefully designed soft-

ware architecture can be used to facilitate the transfer of algorithms between research domains.

Summary of Motivations

The objectives for the development of a software framework can be summarised as follows:

- to allow the development and verification of algorithms on multiple robot platforms
- to allow the code developed in one research domain to be used in another

1.3 Stance for Humanoid Robots

Stance can be loosely separated into two sub–problems, the first being the simple maintenance of an upright posture without perturbations, called quiet stance. For a robot this is a relatively simple task as it corresponds to maintaining a fixed position which is typically performed by low–level position control loops.

The second, more difficult and interesting problem is perturbed stance, where an upright posture must be maintained despite perturbations. Perturbations come in several forms: they may be internal from the movement of a limb [34], or external through the movement of the support surface [35] or application of force [36, 37]. Furthermore, the perturbations can be either continuous [38] or transient [37].

The particular type of perturbation considered in this thesis is the short application of external force, or less formally: a push or impact. Typically, an impact is characterised by a fast transfer of a relatively large amount of energy. If an impact is of a significant magnitude a humanoid robot can quickly fall over. The speed required for impact–recovery presents difficulties to current perturbation sensing techniques.

Limitations of Existing Perturbation Sensing

The Zero Moment Point (ZMP) [39, 40] is frequently used as an input to a stance controller for a bipedal robot. The ZMP is the point on the ground where the net moment of the inertial and gravity forces has no component along the horizontal axes. A controller maintains the ZMP such that it is inside the convex hull of the supporting feet, for example see [41, 42].

By definition the ZMP must remain inside the support polygon; when the robot is falling, the ZMP moves to the edge of the foot about which the robot is rotating. It gives no indication as to how fast the robot is falling and can no longer be used by a controller [43]. This limitation of the ZMP is particularly a problem for the short impacts under investigation here.

There have been extensions to the ZMP allowing it to move beyond the support polygon, in particular, the Foot Rotation Indicator (FRI) [43], the fictitious ZMP [39] and the generalised ZMP [44]. However, they are difficult to measure outside of simulation and are only valid for flat surfaces [45]. Additionally, the ZMP and related concepts do not extend well to compensatory stepping. More recently, there has been research into learning compensatory stepping responses to perturbations [46, 47, 48, 49].

It appears that some of the difficulty in controlling humanoid robot stance stems from the limited amount of information available to the controller. For example, none of the stability measures described above indicate where on the body the impact occurred. In this thesis we seek to improve the quality of stance controllers through improvements in perturbation sensing. Of particular interest are the quantification of location, strength and direction of the impact. Furthermore, as we are dealing with impacts the information must be estimated as quickly as possible.

Chapter 2 reviews and compares the literature on human postural control with typical controllers for humanoid robots. The most significant difference is the dominance of the proprioceptive sense. Whereas, the robotics community measures the ZMP using foot sensors [50, 51, 52] or using an accelerometer and gyro [53, 54]. Proprioception is the sense that provides information about the position and movement joints. A set of foot sensors is equivalent to the sense of touch in the soles of the feet in humans, and the use an accelerometer and gyro are comparable to the sense of balance provided by the inner ear. Consequently,

a secondary motivation for this research on stance was to explore the utility of the proprioceptive sense in the control of stance.

Summary of Motivations

The combination of the two limitations discussed above specifies the aim of the research on perturbed stance for humanoid robots as follows:

- to detect and measure an external impact using only proprioceptive sensors with sufficient detail and speed to allow the generation of a corrective response

1.4 Walk Optimisation for Humanoid Robots

A walk engine is used to generate the intricate motion patterns required for bipedal walking. Walk engines use a wide variety of methods to produce the motion patterns, ranging from detailed models, to ad–hoc approaches [55, 56, 57, 58]. However, regardless of the type of walk engine, they all have a suite of parameters that can be adjusted to tailor a walk engine to a particular robot.

Walk optimisation is the formal process of adjusting these parameters in a systematic way to maximise the performance of a walk engine. The gait generated by a walk engine using a particular set of walk parameters is evaluated through experiment and its performance quantified using a fitness function. The result, along with the results of previous trials, is then used to generate a new set of parameters to be evaluated. This iterative process is repeated until a stopping criterion is satisfied, which is typically a fixed number of iterations or period of time [59, 60].

Challenges of Walk Optimisation

The high dimensionality of the walk parameter space presents a challenge to optimisation techniques. The number of walk parameters can range from 11 [61] to over 50 [56]. The size of the parameter space prevents the use of an exhaustive search for an optimal set of parameters. The high dimensionality also makes the calculation of the gradient at any point in the parameter space difficult.

The time required to evaluate the performance of a particular set of walk parameters also presents a problem for optimisation. A single walk trial may take up to a minute to complete, especially for omnidirectional walk engines where it is necessary to test the walk in several directions [62, 63]. Furthermore, as the notion of walk stability is of particular importance to bipedal walking, a significant amount of time is required to throughly assess the stability of a walk. The lengthy trial time limits the total number of trials to several hundred, or a thousand at best [64, 59, 60].

The stress placed on humanoid robots during the walk optimisation process also presents a challenge. A poor set of walk parameters will produce an unstable gait which can result in the humanoid robot falling. This can occur quite frequently in the walk optimisation process given that it is evaluating, new, untested walk parameters. This places a large amount of stress on the robot and can quickly result in damage.

A human operator [64, 59] or a harness [65] can be used to prevent the robot from falling. Given the time consuming nature of walk optimisation, intense human supervision should be avoided. Furthermore, a harness can limit and effect the movement of a robot, particularly for small fast–moving humanoid robots. Thus, minimising the number of falls during the optimisation procedure is desirable.

The importance of reducing the stress placed on a humanoid robot during optimisation was highlighted when attempting to perform walk optimisation on early versions of the NAO robots. Given their fragility, the number of falls during the optimisation process needed to be reduced significantly for walk optimisation to even be possible. However, the potential for damage when falling is a general concern with humanoid robots and becomes a more significant problem for full–size robots.

Relative Performance of Algorithms and Fitness Functions

There are numerous examples of walk optimisation algorithms in the literature [66, 67]. However, there are few examples where algorithms are compared on the same hardware with the same walk engine. Kohl and Stone [68] compare four algorithms for quadruped walk optimisation and Faber and Behnke [59] compare two algorithms for walk optimisation on a humanoid robot. In both

cases, values for the parameters which tailor an algorithm to a particular problem are selected arbitrarily. The selection of appropriate parameters greatly effects the performance of algorithms [69, 70], thus the parameters should be chosen rigorously through a meta–optimisation.

Furthermore, the performance of walk optimisation algorithms are typically compared based on the number of iterations required to reach the 'optimal' walk parameters. This approach ignores the additional stress placed on the robot by unsuccessful iterations where the robot falls, which is an important consideration for the suitability of an algorithm for humanoid robot walk optimisation.

In the literature there are also many examples of fitness functions for humanoid robot walk optimisation [67]. The two most common being the speed, for example see [64, 59, 60] and the efficiency, for example see [71, 72, 73]. To the author's best knowledge there have been no comparisons of the performance of fitness functions on the same robot hardware.

In this thesis we seek to provide an accurate comparison of optimisation algorithms and fitness functions for humanoid robot walking. The algorithms are meta–optimised prior to their comparison and the relative merits of several fitness functions are compared on the same humanoid robot. To reduce the stress placed on the robot, the cumulative stress during the optimisation is used in place of the iteration count.

Walk Optimisation and Redundant Fitness Functions

Local optimisers are often applied to the optimisation of humanoid robot walking. They are well suited to the problem because of their fast convergence speed and iterative nature. Typically the optimisation process begins with a walk from a manually selected starting point. However, local optimisers converge to sub–optimal local extrema.

The wide variety of potential fitness functions presents an opportunity to apply the concept of safe redundancy [74]. This concept uses several fitness functions sharing a global extremum to escape local extrema. Each time the optimisation stalls in a local extremum the fitness function is replaced by a similar redundant function.

As part of the research conducted in this thesis we aim to apply safe redundancy to extend a local optimisation algorithm. Thus, overcoming the primary limitation of such algorithms.

Optimisation of Joint Stiffnesses

Every walk engine has a set of parameters which effect the joint trajectory pattern produced by the engine, the *traditional* walk parameters. There is a significant body of literature on human walking that suggests in addition to the joint trajectories, the joint stiffnesses also play an important role [75, 76, 77]. In Chapter 2 we review the relevant literature on joint stiffnesses in humans to highlight its significance. The importance in humans suggests that the performance of humanoid robot walking could be improved through the inclusion of joint stiffness patterns in walk engines.

The research conducted as part of this thesis explores joint stiffness for humanoid robot walk engines. In particular, we expand the traditional walk parameter space to include additional parameters that modify the joint stiffnesses. The stiffness parameters, along with the traditional parameters, are selected through optimisation. The benefits of variable joint stiffness are demonstrated by the improved performance of the walk engines when the stiffness parameters are added to the walk optimisation.

Summary of Motivations

To summarise, the aims of the research on walk optimisation are:

- to compare the performance of optimisation algorithms and fitness functions on a single humanoid robot
- to minimise the stress placed on a humanoid robot during the optimisation process without compromising the performance of the final optimised walk
- to use the redundant fitness functions for humanoid robot walking to improve local optimisers
- to investigate the benefits of variable joint stiffnesses as an addition to trajectories generated by a walk engine

1.5 Summary of Contributions and Publications

The main contributions of this thesis cover three areas: software frameworks for robots, stance for humanoid robots, and walk optimisation for humanoid robots. The contributions of the thesis to each area are:

Software Framework

- a cross–robot software framework that supports six different robot platforms and allows code sharing between different research topics

Stance

- a proprioception based impact perception system capable of quickly estimating the location, strength and direction of impacts to the upper and lower body of a humanoid robot

Walk Optimisation

- a cross–robot walk optimisation procedure designed to minimise the stress on a humanoid robot
- an improved walk optimisation algorithm using redundant fitness functions to escape from local maxima
- an additional set of walk parameters that vary the joint stiffness as a function of gait phase

1.5.1 Publications

Software

The software framework has been published in [K5] and has also appeared in the NUbot team descriptions and reports since 2009 [78]. The software framework is open–source and has been publicly available since inception at [79].

Standing

The research on stance, presented in Chapter 4, has been the basis of two publications; [K9] and [K1].

Walking

There have been four publications relevant to the walk optimisation presented in this thesis. The first, [K10], forms the basis for Chapter 5 and was an initial investigation into the benefits of joint stiffness. Chapters 6 and 7 are based on [K4] and [K6], respectively. Finally, Chapter 8 is based on [K8].

Publication List

[K1] Jason Kulk and James S. Welsh. Measuring impacts using support vector machines on a standing humanoid robot. In *International Joint Conference on Neural Networks*, 2012.

[K2] Aaron S. W. Wong, Stephan K. Chalup, Shashank Bhatia, Arash Jalalian, Jason Kulk, Steve Nicklin, and Michael J. Ostwald. Visual gaze analysis of robotic pedestrians moving in urban space. *Architectural Science Reviews*, 2012.

[K3] Jason and James S. Welsh. Evaluation of machine learning techniques for walk optimisation on the nao robot. In *Distributed machine learning and sparse representation with massive data sets*, 2011.

[K4] Jason Kulk and James S. Welsh. Evaluation of walk optimisation techniques for the nao robot. In *IEEE Int. Conf. on Humanoid Robots*, 2011.

[K5] Jason Kulk and James S. Welsh. A nuplatform for software on articulated mobile robots. In *1st Int. ISoLA Workshop on Software Aspects of Robotic Systems*, 2011.

[K6] Jason Kulk and James S. Welsh. Using redundant fitness functions to improve optimisers for humanoid walking. In *IEEE Int. Conf. on Humanoid Robots*, 2011.

[K7] Aaron S. W. Wong, Stephan K. Chalup, Shashank Bhatia, Arash Jalalian, Jason Kulk, and Michael J. Ostwald. Humanoid robots for modelling and analysing visual gaze dynamics of pedestrians moving in urban space. In *The 45th Annual Conf. of the Australian and New Zealand Architectural Science Association (ANZASCA2011)*, 2011.

[K8] Jason Kulk and James S. Welsh. Autonomous optimisation of joint stiffnesses over the entire gait cycle for the nao robot. In *Int. Symposium on Robotics and Intelligent Sensors*, 2010.

[K9] Jason Kulk and James S. Welsh. Perturbation sensing for humanoid robots using a multiclass support vector machine. In *Proc. IFAC Symposium on Mechatronic Systems*, 2010.

[K10] Jason Kulk and James S. Welsh. A low power walk for the nao robot. In *Proc. of Australasian Conf. on Robotics and Automation*, 2008.

1.6 Thesis Overview

A brief summary of each chapter in this thesis is presented below.

Chapter 2 reviews literature on human postural control relevant to the design of the impact perception system described in Chapter 4. The chapter also highlights concepts from human walking relevant to the chapters on humanoid robot walk optimisation.

Chapter 3 describes the cross–robot software framework. The framework employs both a blackboard and message–based system for data transfer and hardware abstraction. A class hierarchy is used to maximise code sharing between robot platforms and research projects. The software described in Chapter 3 is used throughout the thesis, in particular, it provides the cross–robot support for the development of the walk optimisation discussed in Chapters 5 to 8.

Chapter 4 presents the impact perception system for standing humanoid robots. The system uses only joint velocities provided by the humanoid

robot's proprioceptive sense. The detection of the perturbation is performed using a threshold detector. The localisation of the impact is determined using a Support Vector Machine, while the strength and direction are estimated using Support Vector Regression Models. The developed system is applied to a typical humanoid robot.

Chapter 5 describes an initial investigation into the benefits of adjusting the joint stiffness individually for each joint. This work was performed on a physical robot with a walk engine whose traditional walk parameters were particularly limited.

Chapter 6 presents a meta–optimisation, performed in simulation, to select the best parameters for three walk optimisation algorithms. A further meta–optimisation is then performed to select the best algorithm, the best fitness function and most suitable parameter space for humanoid robot walk optimisation. To make the results applicable to physical robot hardware we incorporate the stress placed on the robot during the optimisation in the metric for selecting the best combination. The effectiveness of best combination is then verified using a physical robot.

Chapter 7 extends the optimisation algorithm selected in Chapter 6 to make use of redundant fitness functions. Each time the optimisation becomes trapped in a local maxima, the fitness function is replaced by a different redundant function. The extension is verified in simulation and on two different physical humanoid robots.

Chapter 8 extends the work on joint stiffness presented in Chapters 5 and 6 such that the stiffness is a function of gait cycle. The gait cycle is split into four phases with the stiffness specified independently for each phase. Upon optimisation on a physical humanoid robot, the gait–phase dependent stiffnesses show an improvement.

Chapter 9 presents the conclusions of the work and suggests several avenues for further research.

Chapter 2

Review of Human Motion

Contents

2.1 Introduction 18

2.2 Human Quiet Stance 19

2.2.1 Joint Positions for Stance 19

2.2.2 Proprioception for Detecting Perturbations 19

2.2.3 Joint Stiffness for Stance 21

2.3 Human Perturbed Stance 22

2.3.1 Responses to Perturbations 22

2.4 Human Walking 23

2.4.1 An Efficiency–based Fitness Function 23

2.4.2 Joint Stiffness for Walking 24

2.5 Summary 24

2.1 Introduction

This chapter reviews literature on human motion, highlighting the significant differences between human motion and the current state–of–the–art of humanoid robot motion control. There are numerous examples in engineering where looking at natural systems and biology for inspiration has proved worthwhile [80]. This is particularly true for robotics of all types [81, 82], ranging from a robotic cheetah [83] to a robotic salamander [84].

There is a growing body of research where an examination of humans has resulted in improved humanoid robots [85, 86, 87]. In this Chapter we further examine human motion control in the hope that the ideas and principles can improve the standing and walking of humanoid robots. Indeed, we will see in Chapters 4–8 that this thesis provides more examples of the successful application of biologically inspired concepts.

This chapter is organised as follows, firstly we examine quiet stance to investigate which of the human senses are used to detect perturbations while standing. We find that proprioception is generally the most important sense for motion control and is an essential component for the maintenance of stance. The examination of human quiet stance also reveals that the joint stiffnesses in the legs are quite low.

The second section reviews literature on perturbed stance. In particular, we investigate which properties of an external force effect the corrective responses in humans. Here we find that the responses depend on the location, strength and direction of the perturbation.

The final section in this chapter reviews two key features of human gait that are relevant to the optimisation of humanoid robot walking in Chapters 5–8. The first feature considered is the fitness function used by humans to select a gait, where it appears a function based on energy minimisation is employed. The second characteristic investigated is the joint stiffness used during walking. We will see that the stiffnesses are both joint and gait–phase dependent.

2.2 Human Quiet Stance

2.2.1 Joint Positions for Stance

An important consideration for stance control of humanoid robots is the natural stance position employed prior to a perturbation. There is an extensive body of literature on the pose adopted by humans during quiet stance, the key results of which can be transferred to a humanoid robot.

A study of foot position showed that humans place their feet such that their heel centres were 11.4% of their height apart [88]. It is difficult to use such a pose on a humanoid robot due to their relatively large feet. For example, consider the CycloidII robot, given its height of 42cm, an equivalent foot separation would be 3.3cm. However, the CycloidII has feet which are 6cm wide. Thus, a stance as narrow as possible should be used.

The centre of pressure is maintained by humans to be about 5.5cm in front of the ankle joint [89]. As there is very little movement in quiet stance the positions of the centre of pressure and projection of the centre of mass onto the ground are equivalent. Using anthropometric data from [90] we can calculate that the centre of mass is approximately 3.4° in front of the ankle. We can configure a humanoid robot stance to have a similar centre of pressure by rotating the ankle–pitch degree of freedom.

2.2.2 Proprioception for Detecting Perturbations

The human body has four senses which could be used to regulate stance; touch, balance, sight and proprioception. To determine the contribution of each sense, the measurement threshold and the effect on postural sway when each sense is removed were considered. The measurement threshold of each sense can also be used to specify requirements for an equivalent artificial sense for a humanoid robot.

Touch

The literature on human stance suggests that the mechanoreceptors in the feet are capable of measuring the centre of pressure, however, they are not used by the central nervous system. There are over 100 mechanoreceptors

scattered over the sole of the foot capable of measuring pressures ranging from $2\mu N/mm^2$ up to $22mN/mm^2$ [91]. The pressure under an average human foot during stance is approximately $15mN/mm^2$, so the vast majority of receptors in the sole are stimulated during quiet stance. Furthermore, as an ensemble these sensors are capable of measuring the centre of pressure under the foot, both statically and dynamically.

However, Meyer et al. [92] showed that changes in sway before and after sole anaesthesia were imperceptible with eyes open and small with eyes closed. This small impact implies that the centre of pressure, or ZMP, is not the dominant control variable used during quiet stance. This is in stark contrast to the robotics literature where the ZMP is effectively an industry standard.

Balance

There are at least three sources of our sense of balance, the first is well-known; the inner ear. However, renal and vascular graviceptors contribute significantly [93, 94], that is the distribution of blood and the kidneys provide information which contributes to the human sense of balance. This means a person without a functioning inner ear is not a person without a sense of balance.

The measurement threshold of the sense of balance can be calculated from the data available in [95, 96, 97] to range from 0.3° to 1.2° at velocities of $2.5°s^{-1}$. Furthermore, Bringoux et al. [98] showed that at low velocities people could be rotated by 4.5° to 6° before detecting the movement using the balance sense. These values are significantly higher than those experienced while standing quietly, where the angle varies by $\pm 0.35°$ at velocities of $\pm 0.36°s^{-1}$.

Sight

The sense of sight is the first of the four senses that has a sufficient measurement threshold capable of measuring the small movements that occur during quiet stance. Fitzpatrick and McCloskey [99] performed a direct measurement of the visual systems sway measurement threshold and found it to be well below that of postural sway. This ability depends on the visual surround [100] where the distance to the visual target effects both the anteroposterior and lateral sway.

Proprioception

Proprioception is the most important sense for maintaining upright stance, predominantly because of its ability to provide a large amount of parallel information about the position, velocity and acceleration of each joint. It is clear that the measurement thresholds of proprioception are well below that required to observe postural sway [99], and that removal of proprioception from the legs results in a greater increase in sway than the removal of vision [101, 102, 103, 104].

A particularly compelling case for the utility of proprioception is presented by Sacks [105], where the profound effects of complete proprioception loss are described. Even with a great deal of visual concentration the ability to sit, stand, and walk are severely limited. Furthermore, when visual information is also removed, sitting or standing is no longer possible. This demonstrates that touch, balance and vision can not serve as a substitute for the information that was provided by proprioception.

Typically, a humanoid robot will have accurate position sensors. For example, the CycloidII and NAO have angle sensors with measurement thresholds of 0.3° and 0.09°, respectively, which is comparable to the 0.17° threshold in the human ankle.

It has been observed that joint velocity information is the most accurate component of the proprioceptive sense [106]. A humanoid robot is typically equipped with only position sensors hence the velocities are calculated through differentiation. Consequently, the precision of the joint velocity information available on a humanoid robot is not comparable to that of a human.

2.2.3 Joint Stiffness for Stance

The joint stiffness is defined as being a joint's resistance to external movements [107]. The intrinsic stiffness of a joint is the mechanical stiffness provided by active muscle, tendon and connective tissue. The stiffness can be increased through active contraction of muscles.

Studies of the human ankle have shown that the intrinsic stiffness in the ankle is not sufficient to maintain stance [108, 109], being between 64% and 91% of the value required for marginal stability in the forward–backward plane. The ankle muscles are continuously contracting and relaxing to dynamically

maintain stance. The literature suggests that the hip stiffness is comparable to the value required for marginal stability [37, 75, 110].

To put the human stiffness values into perspective, consider the default settings for the CycloidII and NAO. By default the ankle pitch joint on the CycloidII is a little over 10 times stiffer than that of a human. The default settings on the NAO provide over 100 times more stiffness in the ankle than that of a human. This demonstrates that the stiffnesses typically used on humanoid robots are much higher than those used by humans.

2.3 Human Perturbed Stance

Quiet stance is a special case of the more general perturbed stance. The principles that apply to quiet stance discussed in the previous section, equally apply to perturbed stance. In particular, the proprioceptive sense is essential to the control of stance. Recent evidence [111, 112, 113] suggests that information originating in the torso, from both proprioception and balance, plays a vital role in stance, and that many postural reactions are still present in avestibular patients.

There is a large body of literature on human responses to perturbations. We will use this literature to investigate which characteristics of the external perturbations influence the corrective responses. The measurement of the characteristics identified in this section will become the aim of the impact perception system described in Chapter 4.

2.3.1 Responses to Perturbations

Location Dependence

The response to a perturbation is dependent on the location of the point of contact on the body. Rietdyk et al. [37] noted that when sideward perturbations were applied to either the trunk or pelvis, different responses were induced. Therefore the system we develop must be capable of determining the location of the impact on the body.

Direction Dependence

It is clear that the response required to maintain stance differs depending on the direction of the perturbation. One model for the generation of such responses is that of muscle synergies [114, 115]. In this model, muscles are activated in groups with a fixed relative magnitude and phase. A corrective response is then a superposition of a small set of muscle synergies. Importantly, there are orthogonal synergies for perturbations in the forward–backward plane and the left–right plane. A perturbation which occurs in a direction other than these two planes, results in a superposition of responses from each plane. Consequently the measurement of the direction of an impact should form an essential component of an impact perception system.

Strength Dependence

The response required to maintain stance is also dependent on the strength of the perturbation. Meyer et al. [116] showed that perturbations of increasing strength produced response patterns with similarly increasing amplitude. The study suggests that the same response pattern is used for all perturbations in a particular direction, simply scaled to match the estimated strength of the perturbation. Therefore, the strength of an impact forms a crucial characteristic to be measured by the system we develop.

2.4 Human Walking

2.4.1 An Efficiency–based Fitness Function

There is significant evidence in the literature that a human's gait is selected primarily to minimise the energy consumption [117, 118, 119]. In particular, the minimisation of the total energy used per unit distance travelled [120].

The optimisation of gaits for simulated human models is closely related to the optimisation of humanoid robot walking. Essentially, it is the optimisation of a forward walk, in simulation, for an anthropomorphic model. Typically, the fitness function is based on either minimising the difference between motion capture data and the generated gait [121], or the minimisation of the energy

used [120, 122]. The latter case results in walks that are quite similar to that of a human.

2.4.2 Joint Stiffness for Walking

We saw in Section 2.2 that the stiffness of each joint in the leg is different. Additionally, studies on human gait suggest that the stiffness of each joint in the leg may be phase dependent [123]. Furthermore, the stretch reflexes in the lower leg are modulated as a function of gait cycle [76] where the stretch reflex activity effects the effective joint stiffness. An artificial ankle which provides a fixed stiffness is inadequate at fast walking speeds [124], suggesting that a variable stiffness is required. Furthermore, modelling of a human gait can be improved through the selection of optimal gait phase dependent joint stiffnesses [77].

2.5 Summary

The literature on human stance highlights several features which can be applied to improve the standing of humanoid robots. The joint stiffnesses used by humans during quiet stance are quite low and in the case of the ankle, below that required for marginal stability.

Proprioception appears to be the most dominant sense for perturbation detection, in particular, the joint velocities appear to trigger postural corrections. Furthermore, responses to perturbations of human stance depend on the location, magnitude and direction of the external force.

The literature on human walking suggests that an efficiency–based fitness function is employed to select a gait. In addition to joint motion patterns, the literature suggests that humans use joint stiffness patterns while walking.

Chapter 3

The NUPlatform Software Framework

Contents

3.1 Introduction 26
3.1.1 Related Work 27
3.1.2 Architecture Overview 29
3.2 The Blackboard 30
3.2.1 Sensors 31
3.2.2 Actuators 32
3.2.3 Visual Information 34
3.2.4 Jobs 35
3.2.5 Network Information 35
3.3 The Platform 36
3.3.1 NUPlatform 37
3.3.2 NUCamera 38
3.3.3 NUSensors 38
3.3.4 NUActionators 39
3.4 The Software Modules 39
3.4.1 Behaviour 39
3.4.2 Motion 40
3.5 System Configuration 42
3.6 Applications of NUPlatform 42
3.7 Conclusion 44

3.1 Introduction

This chapter presents the cross–robot software framework designed to facilitate cross–robot development and code sharing between research projects. The design and implementation of the software framework was a significant undertaking due to the number of robots the software needed to support and the variety of the tasks required of the robots. The framework has been used to complete the work presented in the remaining chapters of this thesis and has also served as the basis for the NUbot RoboCup soccer team and other research projects, such as the modelling of pedestrians for gaze analysis [125].

In general, software for robotic systems examines incoming sensor data to generate useful actions that are then executed by a set of actuators. The development of the software is a long and expensive operation, typically with contributions from many developers across a diverse range of fields. Given the size of software systems for robots, a software framework which is portable, configurable and maintainable is desirable.

Robots have numerous sensors and actuators, each requiring a driver to communicate with the software system. Each robot has a unique set of sensors and actuators, which in turn requires a unique set of drivers and a unique set of inputs/outputs for the software system. Consequently, software written for a robot is often tied to that particular robot.

Furthermore, a robotic software system is often comprised of several distinct modules, for example, a vision module, a world modelling module, a behaviour module and a motion module. These modules may also be robot–dependant if communication with the drivers is not done through a sufficient hardware abstraction layer.

This chain of dependancies in the transfer of information from the hardware to the high–level software results in a robot–dependent system. However, the chain can be broken by inserting layers of abstraction which implement standard interfaces that are applicable to all robots. In particular, an abstraction layer is required between the software and the robot hardware, and a standard interface is required for inter–module communication.

The comparison of different versions of the same system module is a common task. This task can be accelerated by making the modules hot–swappable,

as a hot–swappable module can be replaced without restarting the entire system. Furthermore, a single robot platform may be required to perform several unrelated tasks each requiring a specific set of modules. Hence, the ability to swap some modules at runtime allows the robot to switch between tasks. For example, the replacement of a behaviour module designed to play soccer, with one designed to perform a human–robot interaction experiment.

In this chapter we develop a software framework that implements a hardware abstraction layer providing a logical robot [126] to the software system that is identical across robot platforms. The logical robot is presented to the software system as a blackboard [127] which is capable of storing a large variety of different sensor and actuator data. In addition to the data, the blackboard stores information regarding the data's validity that enables the software modules to detect and adapt to sensor or actuator faults.

The software framework also makes use of class hierarchies to both simplify the development of modules and drivers, and to enforce standard interfaces between submodules. In particular, the architecture is designed to keep the robot drivers quite thin, minimising the robot dependent implementation.

The remainder of this chapter is structured as follows: firstly a review of existing software frameworks for robots is presented, followed by an overview of the system architecture of the system. Section 3.2 describes the blackboard and message–based systems employed to standardise the transfer information. Sections 3.3 and 3.4 describe the class hierarchies used to standardise the development of robot drivers and the system modules, respectively. Section 3.6 discusses the successful application of the framework to six robot platforms. Finally, 3.7 presents a summary of the conclusions.

3.1.1 Related Work

There are many software frameworks for robotic systems in the literature [27]. The transfer of information between hardware and system modules can be categorised into two classes; those which are message–based and those which use a blackboard.

A message–based approach excels in distributed systems, as a message can be easily serialised and sent over a network. Notable examples of such systems are ROS [128] and YARP [129], both of which provide an excellent framework

for robot independent software. However, the serialisation of the message, and the data copying on either end, adds overhead to the system. Also, the transport of the message through a physical network adds latency, however, on a single processor system shared memory can be used instead.

A blackboard is well suited to a single mobile processor system because of its lightweight. Blackboards are frequently used in the RoboCup soccer domains given the limited processing available on small mobile robots. Both [130] and [131] use blackboards to share information between system modules.

The architecture proposed in this chapter also uses a blackboard, as many of the target robots have limited processing power. The blackboard is similar to that of [130], in that information is grouped into a small set of classes. The blackboard also incorporates an actuator command queue similar to that of NaoQi [30], enabling the storage of motion sequences and animations. However, the blackboard used here has been generalised to encompass a wide variety of sensors and actuators, which allows the support for multiple robot platforms. In addition to the sensor information itself, the validity of the data is also stored in the blackboard, allowing higher level software modules to detect and adapt to sensor and actuator failure.

A hybrid blackboard–message based architecture is proposed in [132] to take advantage of the strengths of both approaches. A similar feature is used in the NUPlatform framework, where commands, called jobs, can either be shared using the blackboard, or serialised and sent through a physical network. Although, the messages used in the NUPlatform are aimed more toward teleoperation than distributed computing.

Few frameworks provide hardware drivers or implementations of common algorithms. Player [133] and ROS [128] are important exceptions, providing a large software base. Both the Player and ROS frameworks are component–based systems, where a robot is assembled from a set of existing software components. The alternative approach is to use inheritance to allow implementation sharing between similar robot platforms. In the NUPlatform architecture we use a structured class hierarchy to minimise the implementation of drivers and modules, and to enforce standard interfaces. This aspect of the NUPlatform is conceptually similar to several software frameworks for robots, in particular, RoboFrame [134].

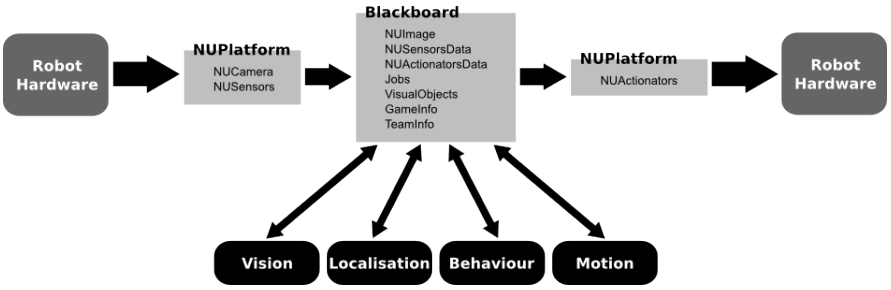


Figure 3.1: An overview of the software architecture and the transfer of information between the hardware and software modules via the blackboard.

Given the dominant target robot platforms for the NUPlatform framework are legged robots, the configuration of the motion system is important. It is common to use a motion manager to select from a list of available motion providers, which provider is going to control the robot [32, 135, 130]. The NUPlatform framework uses the same general principle. However, each limb can be controlled by a separate provider which, for example, frees the arms of a biped to perform other tasks while walking. Additionally, a bridge pattern [136] is used to separate the motion manager from the walk engine allowing the implementation of vastly different engines, in particular, it caters for engines of both biped and quadruped robots.

Finally, of all the robot software frameworks reviewed in this section, none of them have support for the niche set of target robots used here. This was an important consideration for the development of the software architecture.

3.1.2 Architecture Overview

Figure 3.1 outlines the NUPlatform software architecture. The key parts being the blackboard, the system modules and the hardware drivers. The NUPlatform software is open source and has been publicly available since inception at [79]. The entire system has been written in C++ for performance.

The blackboard is central to the system, all of the information transferred between modules is done via the blackboard. The blackboard and its constituent modules are described in Section 3.2.

The NUPlatform object handles the transfer of information between the hardware and the blackboard. It populates the blackboard with sensor data

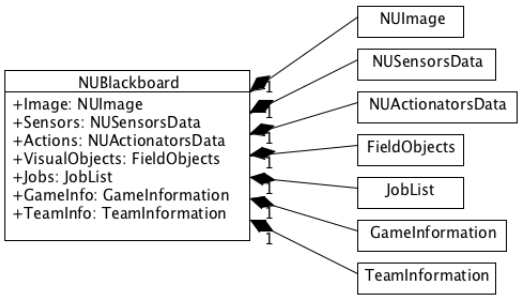


Figure 3.2: A UML class diagram of the blackboard showing the seven constituent parts.

received from hardware and issues the stored commands to the hardware. The NUPlatform object and the associated hardware drivers are described in Section 3.3.

The system modules; vision, localisation, behaviour and motion, also communicate using the blackboard. Sensor data is obtained from the blackboard by the modules. Each module then operates on the data and stores the results of its execution on the blackboard for other modules. The behaviour and motion modules described in Section 3.4 are the two of primary relevance to this thesis.

3.2 The Blackboard

The purpose of a blackboard in a software system is to store data and share it with any modules that require it. Given that the blackboard can be updated and accessed from many threads, it needs to be thread–safe. The blackboard is also used from within real–time threads, so it needs to be very efficient. Furthermore, as it is an object that all developers will use frequently, it should be user–friendly.

The blackboard used in the NUPlatform is shown in Figure 3.2. The information stored in the blackboard is grouped into seven classes based on the source of the information. Each of the classes will be discussed below.

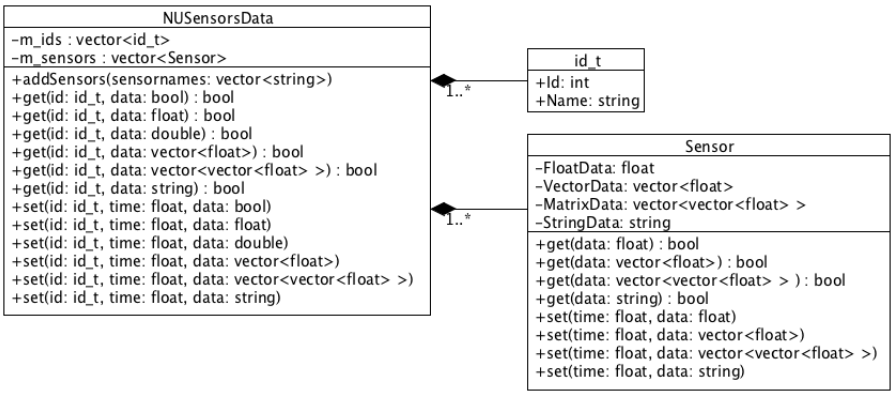


Figure 3.3: A UML class diagram of the NUSensorsData, the class which stores all of the sensor data on the blackboard.

3.2.1 Sensors

The purpose of the `NUSensorsData` is to store sensor data and provide a mechanism for the rest of the system to access this data. A Unified Modelling Language (UML) class diagram of the sensor data store is shown in Figure 3.3.

Figure 3.3 shows that the interface to the sensor data consists of polymorphic `get` and `set` functions. The data produced by any sensor can be reduced to one of the basic data types specified in this interface. This style of interface was chosen to satisfy the requirements of simplicity and efficiency.

The information for individual sensors is accessed using a unique integer identifier. An integer type was used to keep the CPU usage of the system as small possible. However, this approach is not as general, or developer friendly, as using a string identifier. Identifiers for groups of sensors are also provided, for example, an identifier for accessing all of the joint angle sensors.

The data for an individual sensor is initialised as being invalid. When the data is updated using the `set` function it becomes valid. Consequently, for a sensor which is never updated, such as a sensor that is not present on a particular robot platform, the data will remain invalid and the `get` function will always indicate to the rest of the software that the sensor is not available.

Furthermore, it is possible for sensor data to be invalid even on a robot platform that has the particular sensor. The data may become invalid because

of a hardware fault, such as the loss of communication with a single sensor, or by design, such as the kinematically calculated camera height for a humanoid robot becoming invalid when the robot is no longer on the ground.

As an example, consider the measurement of the orientation of the torso of a humanoid robot. The use of accelerometers and gyrometers placed in the chest is the preferred method for measuring the orientation. However, if such sensors either fail, or are not present on a particular platform, the orientation can still be calculated using the kinematic chain of the supporting leg. The result is the software can adapt to the particular robot platform, in real–time, making the system more robust.

3.2.2 Actuators

The purpose of the `NUActionatorsData` is to store commands produced by the software system for the robotic hardware. The command store shares several properties with the sensor data store, and in fact inherits from the same base class. A UML class diagram is shown in Figure 3.4.

Conceptually, a command consists of data describing an action and a time that the action should be performed. This is encapsulated by the `ActionatorPoint` object where the data accepted by hardware actuators is stored along with the timestamp at which the action should be executed.

To simplify the higher–level software modules the `Actionator` object is used to store a queue of `ActionatorPoint` objects. The queue is sorted based on the timestamp for each command with interpolation performed by default on numeric data between consecutive commands. Being able to queue commands makes implementing sequences of actions straightforward, for example, a motion script for a joint or an animation for an LED can be stored in an `Actionator`.

Similar to the sensor data, a single actuator or a group of similar actuators can be addressed using a unique identifier. In the case where a component of the robot is both a sensor and an actuator, for example, a servo motor which provides the ability to both move and sense the position of a link, the same identifier can be used for addressing either the sensor or actuator.

Figure 3.4 shows that the interface to the command store consists of a single polymorphic `add` function. The polymorphism covers both the type of

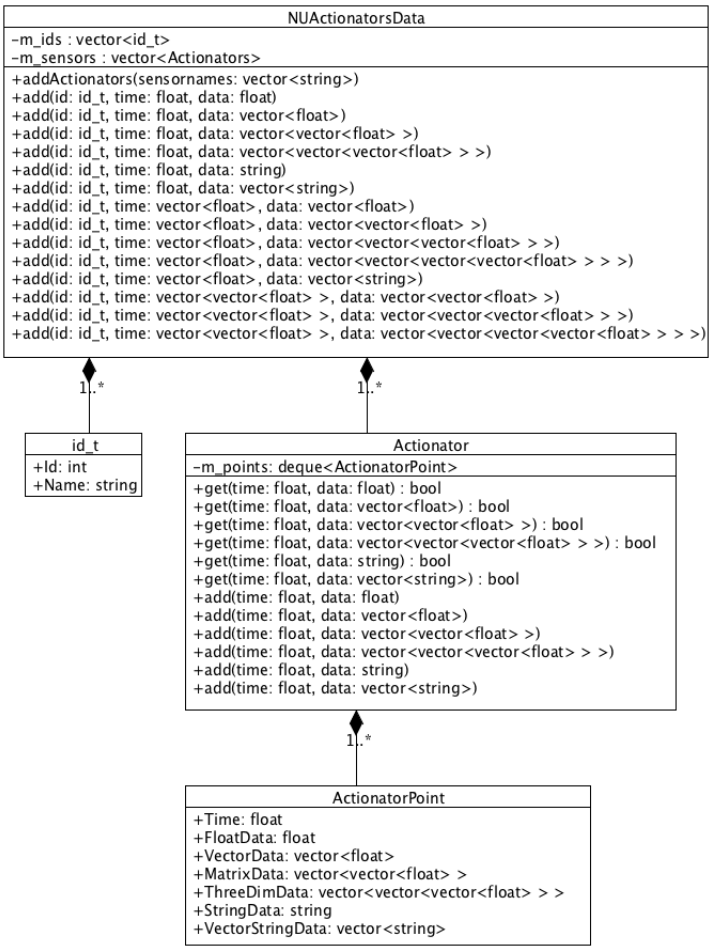


Figure 3.4: A UML class diagram of the NUActionatorsData, the class which holds all of the actions to be issued to the robotic platform.

data to be added, and the type of sequence used to specify the timestamps for the commands.

To understand how the interface works, consider the following function signature as an example

`add(id_t id, vector<double> time, vector<float> data)`.

There are several ways a user might wish to use such a function. If the `id_t` addresses a single actuator, then the timestamps, `time`, and command data,

`data`, will be formatted as

$$[time_0, time_1, \ldots, time_N]$$
$$[data_0, data_1, \ldots, data_N].$$

In this instance, it is clear the user is specifying a sequence of commands to a single actuator.

However, if the `id_t` addresses a group of actuators then the intention of the user is ambiguous. If the timestamps and command data are formatted as

$$[time_0, time_1, \ldots, time_M]$$
$$[data_0, data_1, \ldots, data_M],$$

where M matches the number of actuators in the group, the user is applying a single command to each actuator with a unique timestamp. If the timestamps and data are formatted as

$$[time_0, time_1, \ldots, time_M]$$
$$[data_0, data_1, \ldots, data_L],$$

where $L \neq M$, then the user is applying the same vector with a different timestamp to each actuator in the group. By applying logic to the interpretation of commands specified by the user, the ambiguity in the command can be removed.

A similar approach to the one described in the example above is applied to each of the `add` functions. In the event that a command does not match any of the possible formats, the command is discarded, and the user is alerted that their command was incorrectly formatted.

This approach was chosen to keep the interface as simple as possible. From the perspective of higher–level software modules there is only a single `add` that is used to place any sort of command for the hardware on the blackboard.

3.2.3 Visual Information

Note that the research presented in subsequent chapters of this thesis does not require a robot vision system. However, a robot's vision system is the

primary sense used to perceive its surroundings and is consequently extremely important in other applications, such as RoboCup. A brief description of visual information is included here for completeness.

The visual information is stored in the blackboard in two objects; the `NUImage` and the `FieldObjects`. The `NUImage` stores the image data as YUV422 from the robot's vision sensors. Relevant settings used by the vision sensors at the time the images were captured, such as the resolution, exposure and hue, are also stored in this object. The raw YUV422 image is stored to avoid expensive colour conversions.

The `FieldObjects` stores the visual information extracted from the images after object detection. Each detected object stored in the `FieldObjects` contains its relative position and velocity. This object serves as the primary source of information for the world modelling.

3.2.4 Jobs

The purpose of the `JobList` in the blackboard is to store jobs that are to be executed by a software module. This is distinct from the `NUActionatorsData`; jobs in the `JobList` encapsulate a task at a much higher level, and are to be executed by software modules, not by hardware.

Figure 3.5 shows a UML class diagram of a subset of the available jobs. A class hierarchy is used to share implementation among similar jobs. An iterator is implemented so that each software module can iterate over the jobs in the list, and execute the jobs assigned to it.

For example, consider the `WalkJob`, which controls the movement of the robot. A `WalkJob` is typically generated by the behaviour module. However, a `WalkJob` can be generated on an external system and sent to a robot via a network. In effect, this enables the robot to be easily remote controlled, whether by a human operator, or by another artificial agent. Furthermore, every robot is also capable of transmitting a `WalkJob` over a network, thus enabling any robot to control any other robot.

3.2.5 Network Information

Information received from the network is stored in the blackboard in two objects; `GameInformation` and `TeamInformation`. The origin of the class names

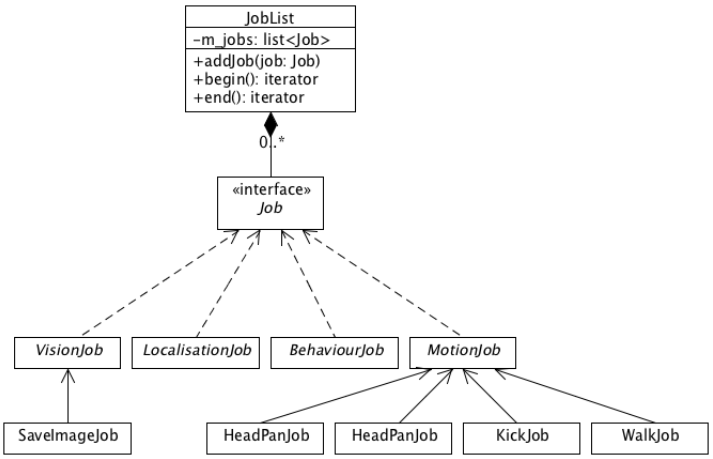


Figure 3.5: A UML class diagram of the `JobList` showing a subset of the class hierarchy that makes up the available jobs.

stem from RoboCup soccer; the `GameInformation` encapsulating the state of the soccer game [137], and the `TeamInformation` encapsulating the state of each of a robot's team mates. However, these concepts can be used in more general domains.

The *game* need not be a soccer match, it could be any sort of task. A referee is used at RoboCup to allow a human supervisor to control the basic behaviour of the robot, such as starting, stopping and pausing. The same referee is used for human supervision of other tasks.

The *team* can consist of robots performing any task. The `TeamInformation` stores the last known position of each robot in the team, as well as the task it was executing.

3.3 The Platform

Figure 3.6 shows the organisation of the robot dependent module of the software system. The platform consists of four submodules; the `NUPlatform`, `NUCamera`, `NUSensors` and `NUActionators`, each of which will be discussed below.

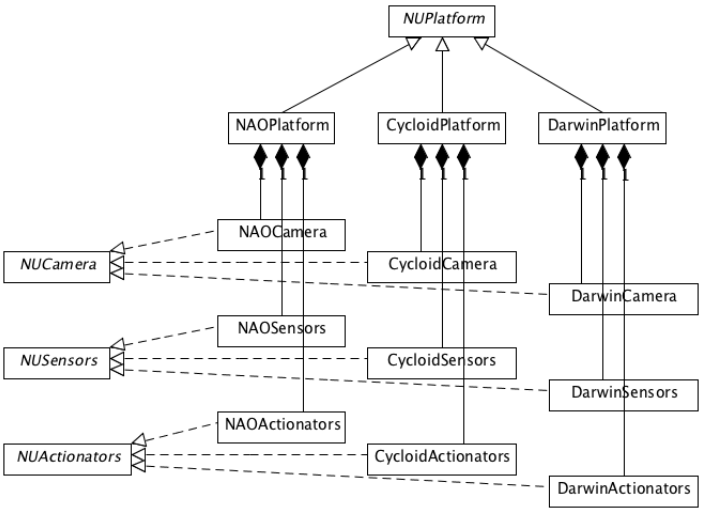


Figure 3.6: A UML class diagram of the NUPlatform showing how the class hierarchy is organised with three example robot platforms.

The flow of information through the four modules is shown in Figure 3.7. The `NUCamera` and `NUSensors` encapsulate the input sensors of the robot, while the `NUActionators` encapsulates all the actuators. These three objects isolate the high–level software modules from the robot hardware. Furthermore, the `NUPlatform` encapsulates the underlying operating system. Consequently, the high–level software modules can be made both robot and operating system independent.

3.3.1 NUPlatform

The `NUPlatform` provides a robot and operating system independent interface for system calls. This includes functions to access the time, threading, and networking of the underlying operating system, as well as providing functions regarding a robot's identity. The class also contains the robot dependent camera, sensor and actuator modules.

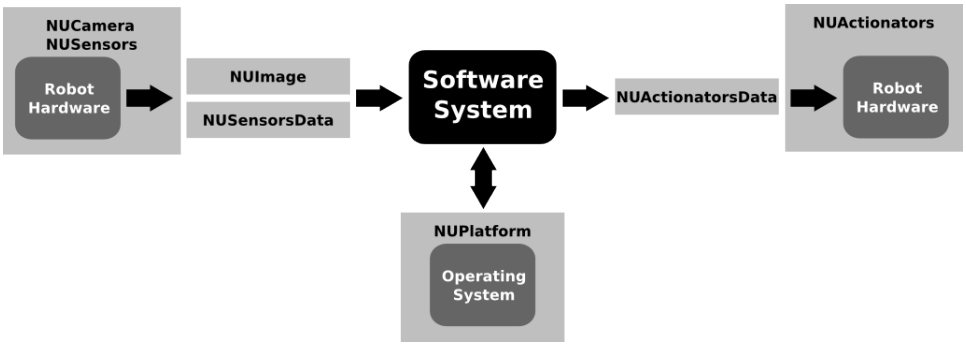


Figure 3.7: An overview of the transfer of data between the software modules and hardware using the NUPlatform framework.

3.3.2 NUCamera

The `NUCamera` provides an interface to the vision sensor of the robot. As noted previously, the vision system of the robot is not required by the work presented in this thesis, however, it is included for completeness and is relevant to the framework's application to other projects. The `NUCamera` has the simple purpose of copying the raw image to the `NUImage` on the blackboard. A simple interface to modify the camera settings is also provided by this class.

The implementation of the class itself is robot dependent. The implementation may be inherited from a generic `NUOpenCVCamera` or `NUV4LCamera`, or it may be robot specific, as is the case with the `NAOCamera`. The implementation is separated from the `NUSensors` as vision processing requires a large amount of memory and processing power. The separation allows specialisations to reduce the overhead of storing the image in the blackboard.

3.3.3 NUSensors

The primary role of the `NUSensors` module is to copy data produced by hardware sensors into the `NUSensorsData` on the blackboard. Before copying the data the `NUSensors` module converts the data into the appropriate format to be stored on the blackboard. The formatting includes both the reduction of the data to one of the accepted types as well as the necessary scaling and ordering to ensure unit and sign conventions are preserved.

The secondary role of the `NUSensors` is to calculate soft sensors and select the best sensor readings to provide a particular sense. For example, consider the orientation of the torso of a humanoid robot. The orientation may be provided by an IMU, in which case this sensor is used. However, the hardware may only have accelerometers and gyrometers, in which case the orientation needs to be calculated. Furthermore, the orientation of the torso can be calculated using the kinematic chain of the supporting leg. When both accelerometers and kinematics are valid, a Kalman filter is used to fuse the information together. However, when there is a sensor fault with accelerometers, or the robot is not on the ground, only the single valid sensor readings are used.

3.3.4 NUActionators

The purpose of the `NUActionators` is to copy the commands, stored in the `NUActionatorsData` on the blackboard, to the actuators. Like the `NUSensors`, the data stored on the blackboard needs to be converted into the proper format expected by the hardware. This process, along with the actual data transfer, is robot dependent.

3.4 The Software Modules

3.4.1 Behaviour

The goal of the behaviour software module is to provide task orientated behaviour for a robot. The behaviour required of a robot is very specific to a target application, and the target applications may be vastly different. To provide behaviour for a wide variety of tasks the system outlined in Figure 3.8 is used.

The system has a single `Behaviour` class which serves as a manager, allowing the selection of a `BehaviourProvider` to implement the task orientated behaviour. The selection of an appropriate behaviour can be done at compile–time, using a button interface while the robot is operating or via a network. The latter approaches are extremely useful when using robots in the field.

The online behaviour switching is done at the beginning of a behaviour cycle. The `Behaviour` class checks if another behaviour has been requested. If

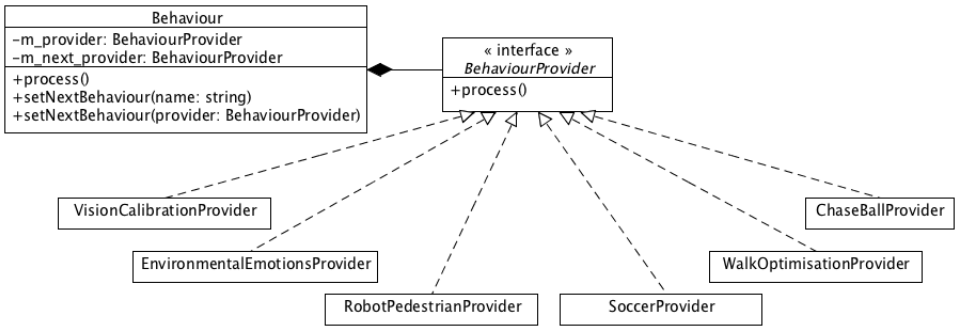


Figure 3.8: A UML class diagram of the Behaviour system showing a subset of the available behaviour providers.

so, the new behaviour is created, set as the current behaviour, and then the previous behaviour is terminated.

The design of the behaviour module in this way effectively reduces the implementation of task and environment specific code to the implementation of a single robot behaviour. For example, the walk optimisation presented in Chapters 6–8 is implemented as a single behaviour alongside a soccer playing behaviour for RoboCup, and many other behaviours for other research projects.

3.4.2 Motion

The motion system provides the robot with a means to move around in its environment. Figure 3.9 shows an overview of the modules that make up the motion system. The general principle behind the system is to have a motion manager select which motion providers should be running at any given time from a list of providers capable of controlling the robot.

The `NUMotion` class is the motion manager. The motion manager divides the joints of the robot into three groups; the head, arms and legs. The manager selects a motion provider for each group of joints where the same provider may be used to control multiple groups if desired. For example, a walk engine is a motion provider that specifies trajectories for both the arms and legs. Usually it will be the active provider for both the arms and legs, however, it is possible for the motion manger to use a different provider to control the arms while the walk engine continues to control the legs.

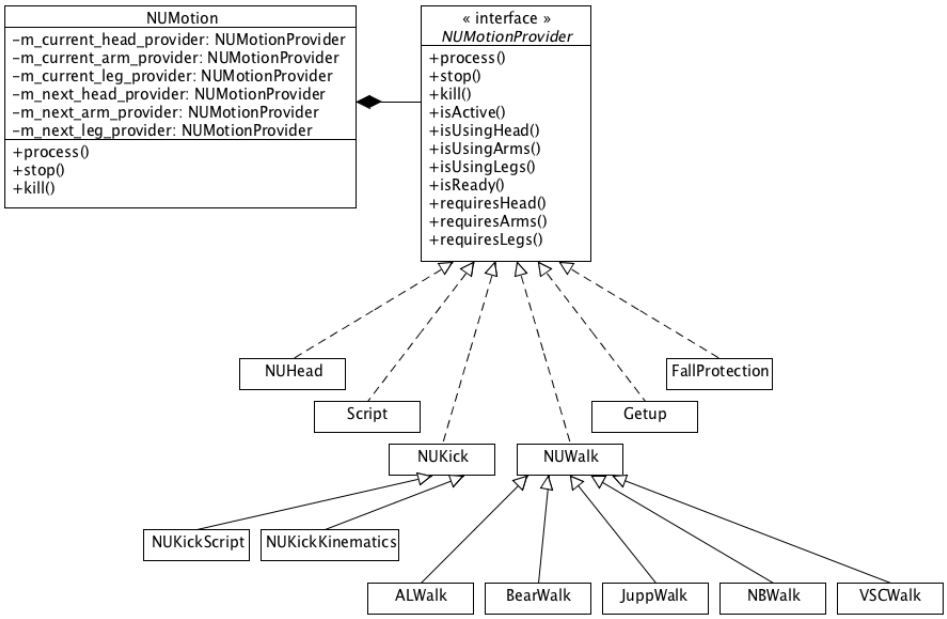


Figure 3.9: A UML class diagram of the Motion system showing the hierarchy of motion providers.

The manager decides which provider to execute based on information in the `NUSensorsData` and the `JobList`. For example, the sensor data is used to determine when the `FallProtection` and `Getup` providers should interrupt the current motion providers. The jobs provide a much smoother transition between providers, waiting for one provider to finish before starting the next.

The motion modules need to be suitable for each of the target robot platforms. The `NUHead` provider controls the motion of the head in a robot independent manner, providing an interface to perform common tasks such as panning and tracking an object. In the case of the `Script`, `Getup`, and `FallProtection` providers, robot dependent configuration files are used to tailor the motions to a specific robot.

The `NUWalk` provider is the most difficult provider to port to each robot platform given the vastly different methods of robot locomotion. A bridge pattern [136] is used to separate the motion manager from the walk engine implementations. It is desirable to use the same walk engine on different

robots [32], however, this is not always possible. For example, a walk engine may only run on a single robot, such as Aldebaran Robotics' walk engine for the NAO, or the robots are too dissimilar, such is the case between wheeled and legged robots.

The `NUWalk` provider selects the appropriate engine to be used with each platform. In the case that multiple engines can be used, the selection is left to the user, and robot specific walk parameters are used to tailor an engine to a particular robot.

The design discussed in this section was vital to the success of the walk optimisation on the three different humanoid robots in Chapter 7. Each of the three robots required a different walk engine, this modules makes that possible.

3.5 System Configuration

The software system is configured using CMake [138]. Each robot platform has a configuration file specifying the required external libraries and platform dependent source files, as well as default values for miscellaneous configuration variables. The result of this is that the system can be built for a particular target using simple commands like `make NAO` or `make Cycloid`.

The user is also able to configure many aspects of the build. In particular, the user can select which system modules to include in the build, that is, the user is able to select whether vision, localisation, behaviour and motion should be compiled. In the instance where multiple implementations of the same module are provided, the user is able to select which implementation to use. For example, the particular walk engine to compile can be selected.

3.6 Applications of NUPlatform

There are currently six platforms supported by the NUPlatform software framework. This includes the four physical robots shown in Figure 3.10, three bipeds and one quadruped. In addition to these robots the framework also supports the Webots simulation package [139], and a generic Webcam. The amount of platform specific code is quite small, at approximately 500 lines per supported platform.

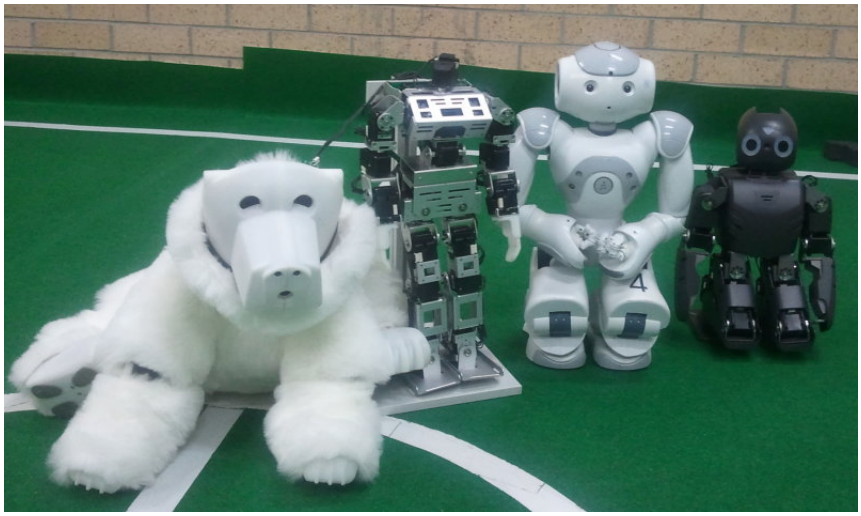


Figure 3.10: The physical robots currently running the NUPlatform. From left to right; the HyKim [28], a modified CycloidII [29], the NAO [30] and the DARwIn [31].

The Webots and Webcam platforms have also been successfully used under the Linux, Mac OS-X and Windows operating systems. However, all of the physical robots run Linux.

One of the primary goals of the work presented in this thesis was to develop algorithms and techniques that could be verified on multiple robot platforms. The software framework presented in this chapter makes such development possible. The CycloidII is used in Chapter 4, the DARWIN-OP is used in Chapter 7, and the NAO is used in Chapter 5 and Chapters 6–8. The Webots simulator is used extensively in Chapters 6 and 7.

The software framework has also be used in a range of projects outside the work presented in this thesis. The first major project was the NUbot's RoboCup soccer team [140] where the framework was used on the NAO in the SPL between 2010 and 2012, and on the DARWIN-OP in the Humanoid league in 2012. The generic Webcam platform was also used under this project for vision development.

The other major project to use the framework was the design of urban spaces through pedestrian analysis [125], where robot pedestrians were used

as an intermediate step between simulation and real–world experiments. This project made use of both the NAO and the HyKim.

3.7 Conclusion

The NUPlatform software framework provides a robot and operating system independent framework for the development of robot software. In particular, a robot independent method for the transfer of information from hardware to high–level modules is provided.

A structured class hierarchy is used to minimise the amount of robot dependent implementation. The class hierarchy also allows the interchange of different implementations of the same modules.

The NUPlatform's flexibility has been demonstrated through its application to six different platforms, including four different physical robots. The framework has also been used in several different projects to provide vastly different robot behaviours.

The software framework described in this chapter has been used as the basis for the implementation of the software required for each subsequent chapter in this thesis. In particular, the framework excelled in chapter 6 where the same walk optimisation technique was applied to three different walk engines on three different humanoid robot platforms. Furthermore, the software framework has served as a basis for the NUbot's participation in the RoboCup soccer competitions from 2010. This included the transition from the NAO to the DARWIN-OP in 2012.

Chapter 4

Impact Perception for a Standing Humanoid Robot

Contents

4.1 Introduction 46
4.1.1 Review of Related Work 46
4.1.2 System Overview 47
4.2 Equipment and Data Collection 48
4.3 Detecting a Perturbation 53
4.3.1 An Optimised Threshold Detector 53
4.3.2 Discussion of Detection Results 55
4.4 Perceiving the Location of a Perturbation 58
4.4.1 Classification of Location Using an SVM 58
4.4.2 Discussion of Classification Results 60
4.5 Estimating the Direction and Strength of a Perturbation 63
4.5.1 Estimation of a Perturbation Using SVR Models . . 63
4.5.2 Discussion of Estimation Results 64
4.6 Effect of Low Joint Stiffness on Performance 66
4.7 Conclusion 67

4.1 Introduction

In this chapter we develop a system that uses only the proprioceptive sense of a standing humanoid robot to determine the location, direction and strength of external impacts. The system consists of an optimised threshold detector for the detection of impacts, a Support Vector Machine (SVM) for the classification of the location of impacts, and two orthogonal Support Vector Regression (SVR) models for estimating the direction and magnitude.

We saw in Chapter 2 that the motion control system of a human relies heavily on the proprioceptive sense. In particular, proprioceptive sensors trigger postural corrections in human stance. Humanoid robots are equipped with a good proprioceptive sense. The servo motors within a humanoid robot provide information about the movement of each joint. A typical humanoid robot has many degrees of freedom in the arms and legs, each of which has a servo motor with a high precision position sensor producing measurements at approximately 100Hz. This provides a large amount of precise information, with a low latency, and hence forms a good proprioceptive sense.

In the literature, the calculation of the ZMP is a common use of the proprioceptive information on humanoid robots. Chapter 1 discussed the limitations of the ZMP for sensing external impacts and Chapter 2 provided evidence that the ZMP is not used by humans to maintain stance. Thus, an alternative use of the proprioceptive sense is required.

Furthermore, Chapter 2 illustrated that the location, direction and strength of perturbations influenced the corrective responses in humans. Therefore, we focus on measuring these properties using the proprioceptive sense.

The perturbation sensing provided by the system developed in this chapter could then be used to generate a better response to perturbations. Additionally, a sense of 'touch' would also be provided without the need for a tactile sensor skin, allowing the robot to better interact with its surroundings.

4.1.1 Review of Related Work

There have been no attempts in the literature to characterise perturbations to stance using proprioception. However, the problem is similar to that of detecting collisions, which has been studied previously. For example, collision

detection during walking for both quadruped [141] and humanoid [142] robots has been considered.

The approaches described in the literature are similar, a detector is created by training a model on the expected proprioceptive sensor readings [141, 142, 143, 144]. A perturbation is then detected as a large deviation from the expected sensor readings. The model may be an SVM [144], a Neural Network [142], or other expert system [141, 143].

Importantly, the aforementioned work does not quantify the external influence. It is typical for a collision detection system to only produce a binary output [141, 143], where the robot is either perturbed or not. There is no estimate of the strength or location of the perturbation. The detection delay for the two approaches [141, 143] is approximately 0.3s, as both rely on the observation of positive detections over several motion frames.

A binary output for each joint is produced by [142]. This information could be used to localise the perturbation, nonetheless, neither the magnitude nor direction of the perturbation are estimated. The detection delay for this approach is at least 50ms, however, accurate truth data for the perturbations was not collected.

4.1.2 System Overview

The system we propose consists of three components; the detector, the classifier and the estimator. The detector produces a simple binary output as to whether the robot is currently perturbed or not. To detect the perturbation, a norm is applied to the joint velocities, if the norm exceeds a threshold, the robot is considered to be perturbed. The detector is described in detail in Section 4.3.

The problem of determining the location of a perturbation is posed as a classification problem. The location estimate is discretised into 16 categories; 8 on the upper body, and 8 on the lower body. Essentially, the perturbation location is discretised to individual limbs. This approximation is reasonable because a humanoid robot is made of rigid limbs and motors with only a single degree of freedom. The application of force anywhere along such a limb produces a similar effect, since the limb effectively acts like a lever.

The classification itself is performed by a multi–class SVM [145] on the joint velocity samples after the initial detection of the perturbation. The parameters

for the SVM and radial basis function are optimised to maximise classification success. The classifier is outlined in Section 4.4.

The final component of the system is the estimator, which determines the direction and strength of the perturbations to the upper body. The estimator also uses the joint velocity samples after the initial detection. It is composed of two SVR models, one for the forward–backward plane, the other for the left–right plane. The parameters for the two SVR models are optimised independently to minimise the mean–squared error. The estimator is discussed further in Section 4.5.

In order to train the SVM and SVR models, truth data for the perturbations was required. The force applied to the robot during a perturbation was measured using 16 pressure sensors placed on the upper and lower body of the robot. These sensors provide truth data in regards to the timing of the perturbation, which is used in Section 4.3 to optimiser the detector. The sensors also provide the location and strength of the impact which is used to train the SVM in Section 4.4 and the SVR models in Section 4.5. The collection of the data is described in the next section.

4.2 Equipment and Data Collection

The humanoid robotic platform used for this work was a CycloidII [29]. The CycloidII was used because of its availability and it being representative of a typical small humanoid robot constructed from servo motors. It is 41cm in height, weighs 2.8kg and has 23 degrees of freedom; 6 in each leg, 4 in each arm, 2 in the torso and 1 in the head. The robot has been modified to accommodate a Geode CPU running Linux and the NUPlatform software architecture. The software system was configured to calculate the joint velocities by differentiating the positions provided by the servo motors at 50Hz. An α–β filter [146] was applied to the derivative to reduce noise.

Prior to a perturbation the robot is standing quietly in the position shown in Figure 4.1. The position was configured to mimic the identified features of human stance described in Section 2.2.

Section 2.2 also highlighted the low joint stiffnesses of humans during quiet stance. To achieve comparable stiffness on the robot the joint stiffnesses were selected to be as low as possible, whilst retaining the ability to stand without

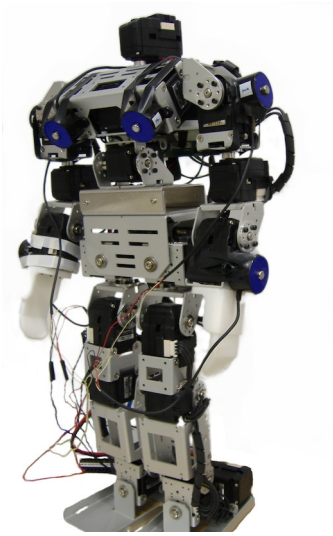


Figure 4.1: The standing position of the CycloidII prior to perturbation. The knees are locked straight and the arms are left to swing freely. The orientation was selected such that minimal torque is required at the ankles and hips.

active changes to the servo inputs. This procedure results in stiffnesses close to that required for marginal stability, which was seen to be the case for humans. The controller gain for each servo motor was used to specify the stiffness for each joint.

The robot stance with anthropomorphic positions and stiffnesses was very compliant and allowed the robot to move significantly during perturbations, as opposed to simply rotating about an edge of a foot.

Force sensors were placed at 16 locations on the robot to collect the truth data of the perturbation. The sensors were placed at 8 locations on the upper body, and 8 locations on the lower body, as shown in Figure 4.2. The force sensors were connected to an external computer using a data acquisition card and recorded at 75Hz.

Figure 4.3 shows two examples of the force measured during a perturbation. The perturbation consists of a force lasting approximately 0.25s. The impulse of the impacts ranged from 0.3Ns to 2.0Ns, with a mean of 0.9Ns. Experiments involving similar perturbations have been performed on humans to investigate their posture control [37], where the impulse of impacts ranged from 13Ns to

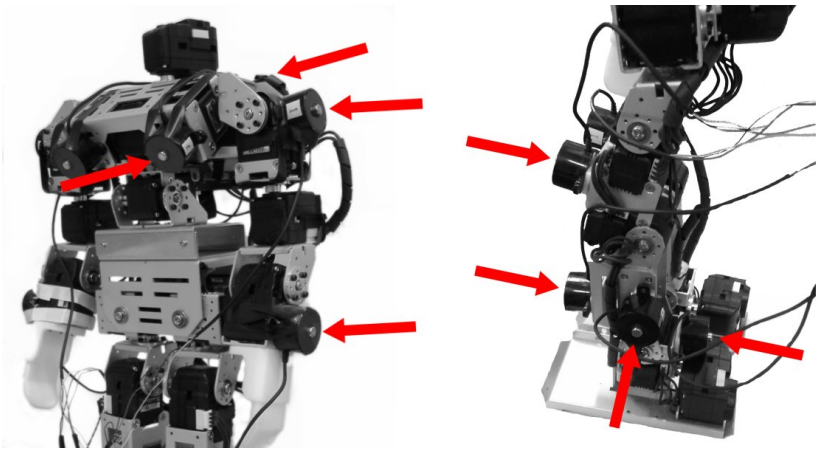


Figure 4.2: The placement of the force sensors used to collected training data. Sensors were also placed on the right side in the same configuration as those shown in the figure.

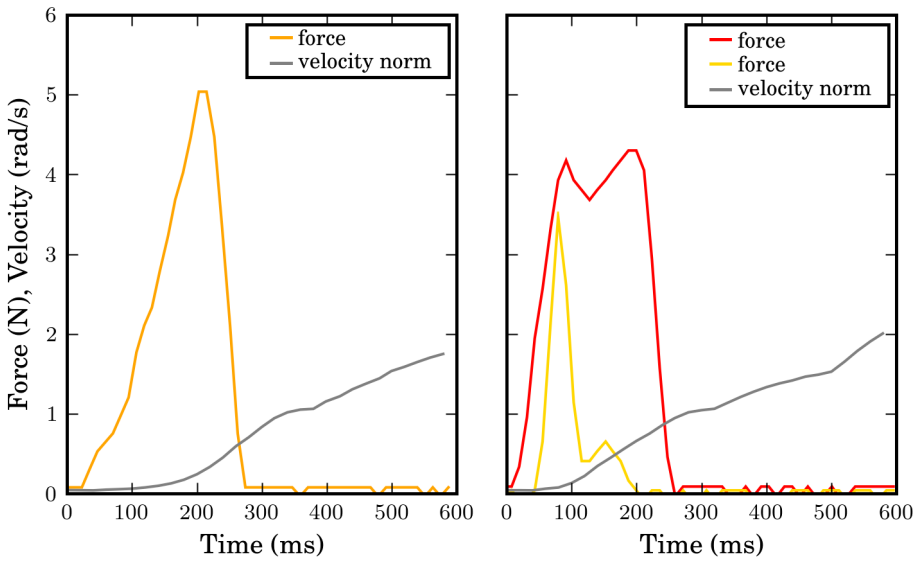


Figure 4.3: Two impacts showing the force measured by the sensors attached to the robot. The first impact is applied to a single sensor, a member of the orthogonal perturbation–set. The second impact is applied to two orthogonal sensors and is a member of the omni–directional perturbation–set.

31Ns. The impulses applied to the robot, scaled to account for its smaller mass, are approximately equivalent to impulses on a human of 7Ns to 50Ns. Thus, the range used here appears suitable for the development of anthropomorphic perturbation sensing. Furthermore, the physical effect of the perturbations on the robot range from it moving slightly, but remaining standing, to the robot falling over quickly. Similar effects were observed in [37] for humans.

Perturbation data where force was applied to a single sensor was collected for the training of the location classifier and strength estimator. Perturbations in this subset are either in the direction of the sagittal (forward–backward) or coronal (left–right) plane. This subset of data will be called the orthogonal perturbation–set.

Perturbation data where force was applied to a sensor in both the sagittal and coronal planes was collected for the training and verification of the strength estimators. Through the application of varying amounts of force in both planes we can emulate perturbations in any direction with any magnitude. This subset of data will be called the omnidirectional perturbation–set.

After each perturbation the robot is manually returned to its quiet stance pose and given a short period to settle before the application of the next impact. The data recording is performed continuously throughout the experiment until all of the perturbations have been applied. This procedure enables approximately 15 perturbations to be captured per minute. The raw data is then processed offline. The manual repositioning is removed from the data and the samples belonging to each perturbation are extracted. The samples between the manual repositioning and the next perturbation are also extracted for selecting the detection threshold.

The dataset collected for the impact perception consisted of 603 perturbations; 270 on the upper body, 220 on the lower body and 113 perturbations of multiple sensors on the upper body. A small selection of the joint velocities and perturbation forces for several perturbations is shown in Figure 4.4.

The direction and magnitude of the perturbations collected for both the orthogonal and omni–directional perturbation–sets are shown in Figure 4.5. Since a humanoid robot is inherently less robust to backward perturbations, and more resilient to forward perturbations, the range of perturbation strength was slightly smaller in the backward direction than in the forward direction.

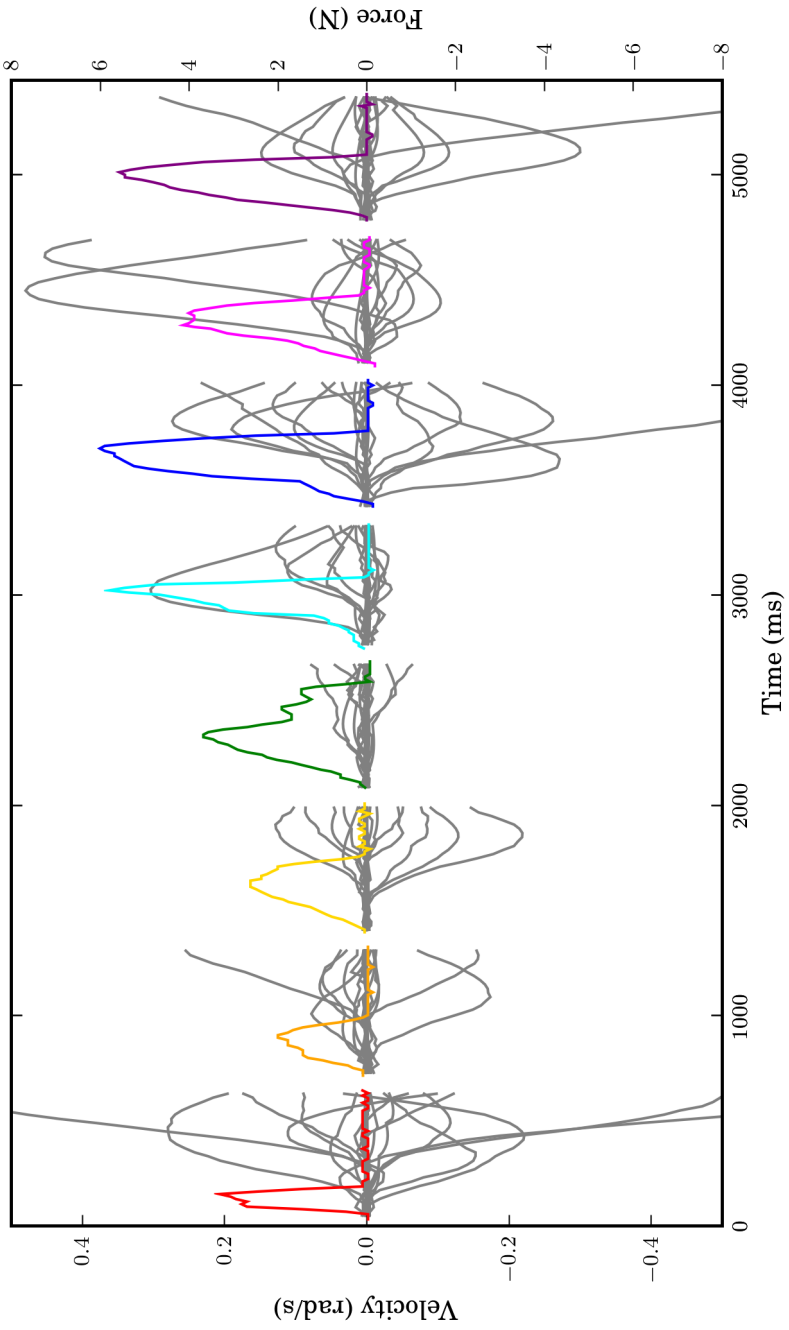


Figure 4.4: A selection of 8 perturbations, each perturbation is at a different location on the upper body. The grayscale lines are the 23 individual joint velocities and the coloured lines are the forces measured by each pressure sensor. Note the discontinuities in the time axis separate each of the perturbations.

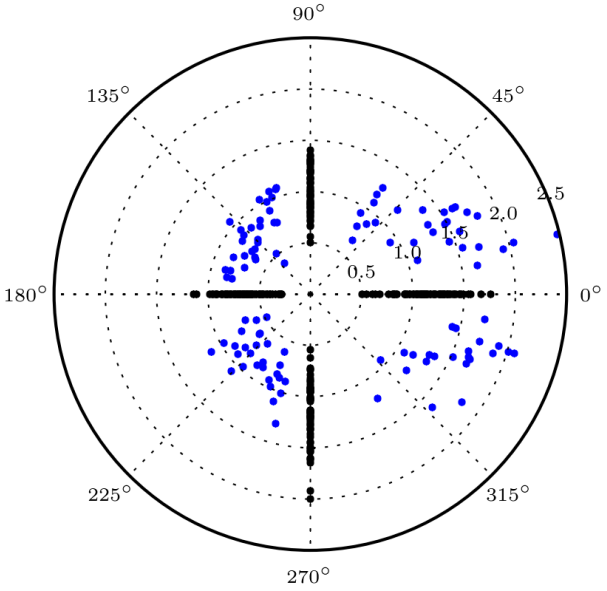


Figure 4.5: A polar diagram of the perturbations. The perturbations applied to a single sensor are black, and lie along the axes. The perturbations applied to two sensors are shown in blue. An angle of 0° represents a forward perturbation, that is a perturbation where the robot was pushed forward from behind. The magnitude of the perturbation is in Newton seconds (Ns).

4.3 Detecting a Perturbation

4.3.1 An Optimised Threshold Detector

To detect whether the robot has been perturbed we use a weighted sum of the rotational joint velocities, $d(\dot{\mathbf{\Theta}}^{\mathbf{k}})$ where $\dot{\mathbf{\Theta}}^{\mathbf{k}} = \{\dot{\theta}_i^k\}_{i=1}^{23}$ is the rotational velocities of each of the 23 joints at the kth time sample. When the sum exceeds a threshold, T, the robot is deemed to have been perturbed.

There are many choices for the function $d(\dot{\mathbf{\Theta}}^{\mathbf{k}})$. The two functions considered here were the weighted Manhattan norm,

$$d(\dot{\mathbf{\Theta}}^{\mathbf{k}}) = \sum_{i=1}^{23} w_i \left|\dot{\theta}_i^k\right| \tag{4.1}$$

and a weighted seminorm,

$$d(\dot{\mathbf{\Theta}}^{\mathbf{k}}) = \left| \sum_{i=1}^{23} w_i \dot{\theta}_i^k \right|. \tag{4.2}$$

There are two aspects of the detector that can be adjusted; the threshold, T, and the weights, $\mathbf{w} \in \mathbb{R}^{23}$. The selection of T is based on the maximum $d(\dot{\mathbf{\Theta}}^{\mathbf{k}})$ that was observed in all of the unperturbed samples collected. T is then set to 110% of this maximum value, resulting in a false positive detection rate of zero over the collected data. Mathematically, the threshold is given by

$$T = 1.1 \cdot \max_k d(\dot{\mathbf{\Theta}}^{\mathbf{k}}), \quad \text{for all } k \in \text{Unperturbed–set}.$$

The weights, $\mathbf{w}$, specify the amount each joint contributes to the detection norm. There inclusion allows the adjustment of the norm to give greater influence to particular joints. The selection of the weights poses an optimisation problem, where the objective is to minimise the average detection time, t_0. Formally, we can write the optimisation problem for finding $\mathbf{w}$ as

$$\min_{\mathbf{w}} t_0$$

$$\text{subject to:} \quad w_i \in [0, 1] \text{ for } i = 1, \ldots, 23.$$

The detection time is calculated as the time between the initial onset of force, as measured by the force sensor, to the time the norm crosses the detection threshold. For each set of $\mathbf{w}$ the detection threshold is recalculated.

Due to the non–convexity of the problem, Particle Swarm Optimisation [147] was used to determine the optimal weights. The dimensionality of the problem was reduced by constraining the weights for joints on the left and right sides to be equal. Several degrees of freedom, one in the chest and two in each arm, do not move during any impacts and were given weights of zero.

The average detection time after optimisation over the entire dataset of perturbations was 138ms and 163ms using (4.1) and (4.2), respectively. The weights selected by the optimiser for both norms are shown in Table 4.1. Note that the average detection time was 175ms when $w_i = 1$ for all i for both (4.1) and (4.2).

Table 4.1: Joint Weights Selected through Optimisation

(a) Weights for (4.1)

Joint	Weight
ShoulderPitch	0
ShoulderRoll	0.40
HipRoll	1
HipYaw	0.71
HipPitch	0.16
KneePitch	0.19
AnklePitch	1
AnkleRoll	0.83
TorsoYaw	0.22

(b) Weights for (4.2)

Joint	Weight
ShoulderPitch	0.29
ShoulderRoll	0.17
HipRoll	0.76
HipYaw	0.41
HipPitch	0.04
KneePitch	0.21
AnklePitch	0.95
AnkleRoll	1.0
TorsoYaw	0.55

In terms of detection time, the weighted Manhattan norm (4.1) outperforms the seminorm (4.2). Thus, (4.1) is used as the detection function in the remainder of this chapter.

4.3.2 Discussion of Detection Results

It is evident from Tables 4.1(a) and 4.1(b) that the weights are similar for both detection functions. The hip roll, ankle pitch and ankle roll joints are the most dominant sources of information used by the detector. This is due to these joints having lower joint stiffnesses than, for example, the hip and knee pitch joints, allowing more movement when a perturbation occurs.

The improvement due to the optimisation of the weights for (4.1) is shown in Figure 4.6. It is apparent that the optimisation is able to significantly reduce the threshold.

The average detection time for perturbations of the upper and lower body were significantly different, being 95ms and 215ms respectively. The perturbations of the lower body are more difficult to detect because fewer joints move as a result of the perturbation. Furthermore, the robot's feet have a tendency to slip easily when perturbations are applied to the lower body. Both of these factors contribute to there being less information available for the detection of impacts.

In some instances the velocity norm is slow to cross the threshold, reducing the performance of the detector. Figure 4.7 shows several impacts where the

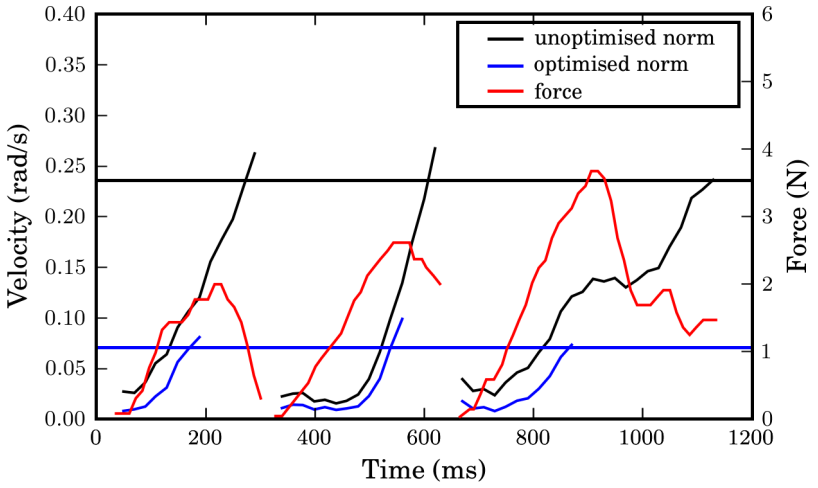


Figure 4.6: The norm (4.1) of the joint velocities before and after optimisation. The thresholds selected before and after optimisation are shown by the horizontal lines. The force measured by the pressure sensor is also shown.

detection delay is much greater than the average. It is clear from the figure that there is a significant delay between the measurement of the force and any resultant motion. This effect is especially prevalent for impacts to the lower body and explains why the detector performs worse for that subset.

There are several causes for the observed measurement delay, the first being the force sensor mountings. The force sensors are secured to the robot using double-sided tape, which is essentially foam. This presents a small delay between the transfer of the force from the sensor to the robot itself. There is also a small amount of backlash in the robot's joints and limbs, this movement can not be measured using the servo motors. Finally, a small delay is introduced by the α–β filter.

It is difficult to compare the performance of the proposed detector to those in the literature. Collision detection is frequently used to modify a robot's behaviour, where the detection delay is not critical to the success of the detector. Consequently, ad–hoc measurements of the detection delay are presented in [141, 142, 143]. The mean detection time of 138ms achieved by the proposed

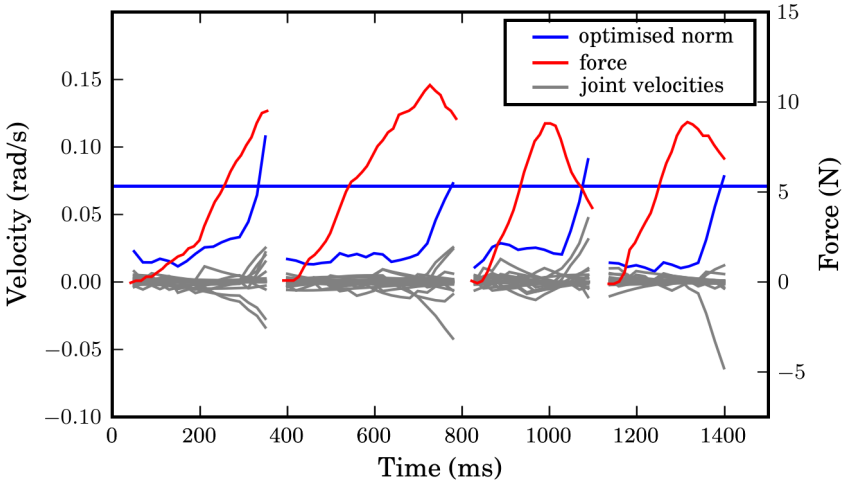


Figure 4.7: Impacts where the detection delay is large. The individual joint velocities are included to illustrate that the delay is a physical delay and not caused by the calculation of the norm.

system is comparable to the delay observed on humans of between 125ms and 200ms [37, 116].

The perturbation detection system proposed in this section could also be applied to other humanoid robots that have accurate joint sensors. The CycloidII has 10-bit joint position sensors that can be recorded at a frequency of 50Hz. This is not a particularly high for a humanoid robot, for example, both the NAO [148] and DARWIN-OP [149] have 12-bit position sensors that can be recorded at 100Hz. The improved resolution and capture frequency should enable a lower latency between the perturbation and its detection.

For the application of the system to another humanoid robot the external force sensors would need to be attached during data collection. The weights for the velocity norm would need to be re–optimised for the new data and robot.

4.4 Perceiving the Location of a Perturbation

4.4.1 Classification of Location Using an SVM

The problem of localising the point of impact can be simplified into a classification problem by discretising the location into a set of possible contact points. We categorise the location into a set of 16 classes, $\mathbf{Y}_\ell$, over the upper and lower body of the humanoid robot, where

$$\begin{aligned}\mathbf{Y}_\ell = \{&\text{FrontRightTorso, FrontLeftTorso,}\\ &\text{RightShoulder, LeftShoulder, LeftArm, RightArm,}\\ &\text{BackRightTorso, BackLeftTorso,}\\ &\text{FrontRightThigh, FrontLeftThigh, FrontRightShin, FrontLeftShin,}\\ &\text{RightShin, LeftShin, BackLeftShin, BackRightShin}\}.\end{aligned}$$

A force sensor was placed at each of these locations as shown in Figure 4.2.

The input vector for the SVM is a vector of joint velocities $\mathbf{x}_k$. The vector is formed from three consecutive samples of the velocity from each joint. Formally the kth vector $\mathbf{x}_k$ is given by

$$\mathbf{x}_k = \{\dot{\mathbf{\Theta}}^k, \dot{\mathbf{\Theta}}^{k-1}, \dot{\mathbf{\Theta}}^{k-2}\} \tag{4.3}$$

where $\dot{\mathbf{\Theta}}^k$ is the kth vector of the velocities $\{\dot{\theta}_i\}_{i=1}^{23}$ from every joint. This forms a 69 dimensional input vector. The joint velocities are also normalised such that the values for each joint range between ± 1.

We use three consecutive velocity samples to provide a *pattern* on which to perform the classification. We note that the three samples span 60ms, whilst the average detection delay is 138ms. Including samples prior to the detection provides a diminishing amount of information, thus further increasing the number of velocity samples per vector does not improve performance.

The orthogonal perturbation–set, described in Section 4.2, was used as the training dataset for the classification. In particular, we use the first five $\mathbf{x}_k$ vectors after the detection of each perturbation for the training of the SVM, as shown in Figure 4.8. We use the first five vectors as we aim to classify

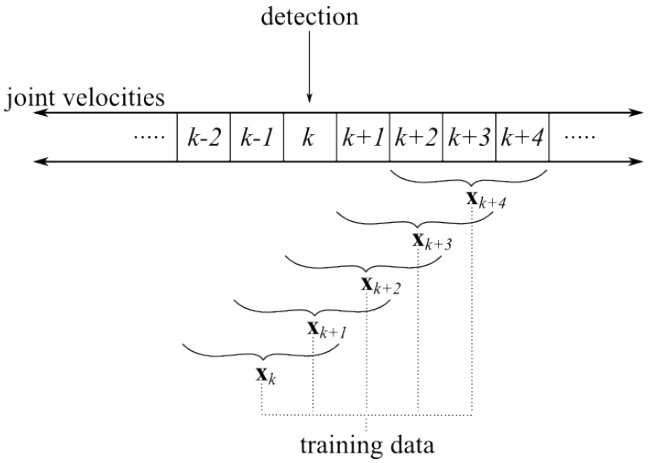


Figure 4.8: A diagram showing the formation of the training data used for classification of the perturbation location.

the location immediately after the perturbation. Each $\mathbf{x}_k$ is assigned a label, $y_k \in \mathbf{Y}_\ell$, using the truth values collected via the force sensors.

A one-against-one multi-class SVM [150] with a radial basis function kernel performs the classification. In particular, the LIBSVM [145] implementation is used. There are two parameters that can be adjusted to improve the performance of the classification. The first is $C > 0$, the penalty parameter of the error term in the primal form [151]

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i$$

$$\text{subject to:} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable. The second is $\gamma > 0$, the parameter of the radial basis function kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$$

where $\mathbf{x}_i$ is the ith sample vector. A grid search was employed to select the parameters C and γ to minimise the error rate of a 10-fold cross validation.

After selection of suitable SVM parameters the error rate of a 10-fold cross

validation was less than 0.2%, with only 4 errors in the 2435 samples. An example confusion matrix is shown in Table 4.2. The small number of errors occur only in impacts on the lower body, in particular, the front left thigh and the front right shin. The classification performs flawlessly on the upper body.

To overcome the small number of errors in the lower body a simple voting system is used. If two consecutive vectors, $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$, are both classified as belonging to the same class, then the location of the perturbation becomes this class. However, if the two vectors disagree, a third vector $\mathbf{x}_{k+2}$ is used to cast the deciding vote. This simple strategy works well here as no perturbation has more than a single classification error.

Overall, the classifier is able to localise an impact to the upper body using only a single sample, thus the average delay is the same as the detection delay, 95ms. The addition of the voting strategy to impacts on the lower body increases the time taken to correctly localise the impact by 20ms, taking the average classification delay from 215ms to 235ms.

4.4.2 Discussion of Classification Results

The computational load of the classification is quite low. A single classification can be performed on the Cycloid's CPU in approximately 1.5ms, comfortably allowing the system to run in real–time.

To gain insight into why the classification is capable of achieving near perfect accuracy, Principal Components Analysis (PCA) was performed on the joint velocity vectors used for classification. Figure 4.9(a) shows that there is a large degree of separation inherent in the data collected for the perturbations to the upper body. Figure 4.9(b) shows that there is a smaller amount of separation of the impacts on the lower body, hence the reduced performance of the classifier on the lower body.

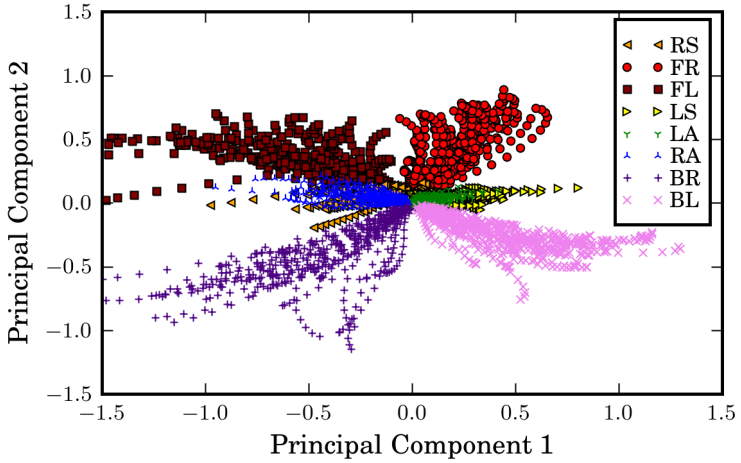
The perturbation locations chosen for classification are quite generic and could equally be applied to most humanoid robots. Furthermore, the location and configuration of each joint is not unique to the CycloidII, most humanoid robots have similar degrees of freedom in the legs and arms. Thus, the movement caused by perturbations at each location should 'look' similar on most humanoid robots, and produce fairly unique motion pattern that can be easily classified. However, for the system to be applied to another robot new data

Table 4.2: Confusion matrix for perturbation location classification

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	190	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	195	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	150	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	171	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	145	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	132	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	168	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	189	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	165	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	**1**	152	0	0	0	0	**1**	0
11	0	0	0	0	0	0	0	0	0	0	118	0	0	0	**2**	0
12	0	0	0	0	0	0	0	0	0	0	0	105	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	145	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	137	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	115

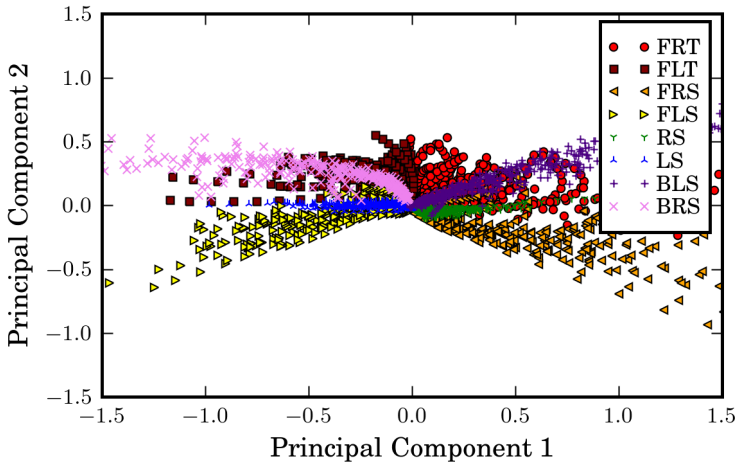
Key:

1:	Front Right Torso	2:	Front Left Torso	3:	Right Shoulder	4:	Left Shoulder
5:	Left Arm	6:	Right Arm	7:	Back Right Torso	8:	Back Left Torso
9:	Front Right Thigh	10:	Front Left Thigh	11:	Front Right Shin	12:	Front Left Shin
13:	Right Shin	14:	Left Shin	15:	Back Left Shin	16:	Back Right Shin



(a) A Biplot for perturbations to the upper body.

RS:	Right Shoulder	FR:	Front Right Torso	FL:	Front Left Torso
LS:	Left Shoulder	LA:	Left Arm	RA:	Right Arm
BR:	Back Right Torso	BL:	Back Left Torso		



(b) A Biplot for perturbations to the lower body.

FRT:	Front Right Thigh	FLT:	Front Left Thigh	FRS:	Front Right Shin
FLS:	Front Left Shin	RS:	Right Shin	LS:	Left Shin
BLS:	Back Left Shin	BRS:	Back Right Shin		

Figure 4.9: Biplots of the first two components found by PCA on the $\mathbf{x}_k$ samples.

would need to be collected and the SVM would need to be retrained to fit the new data and robot.

4.5 Estimating the Direction and Strength of a Perturbation

4.5.1 Estimation of a Perturbation Using SVR Models

In this section we are interested in quantifying both the strength and the direction of a perturbation using only the joint sensor information. The impulse of a perturbation is a commonly used measure of strength in the literature on human stance [116, 37] and is easily measured by integrating the force measurements of a pressure sensor. Therefore, we use the impulse to characterise the strength of a perturbation. The direction of the perturbation is the vector parallel to the ground along which the force was applied to the robot.

To simplify the estimation of the magnitude and direction we decompose the force into components in the orthogonal sagittal and coronal planes, m_x and m_y, respectively. The two components are then used to form a total magnitude and a direction via

$$\begin{aligned} \text{mag} &= \sqrt{m_x^2 + m_y^2} \\ \text{dir} &= \arctan \frac{m_y}{m_x}. \end{aligned}$$

Formally, we need to fit two models; one for the impulse in the sagittal plane, $M_x : \mathbf{x}_k \mapsto m_x$; and another for the impulse in the coronal plane $M_y : \mathbf{x}_k \mapsto m_y$. An individual SVR model is used to represent each map, in particular, the ν-SVR [152] with a radial basis function kernel is used.

Section 4.2 described the collection of orthogonal and omnidirectional perturbation–sets, which were shown in Figure 4.5. The orthogonal perturbation–set was used for training, and the omnidirectional perturbation–set was used for testing and verification. Essentially, the SVR models were trained on perturbations where force is applied purely in the sagittal and coronal planes and then tested on perturbations that have forces applied in mixed directions. Only the perturbations to the upper body were included because of

the difficulties associated with collecting the lower body dataset as discussed in Section 4.3.

The truth value for the impulse of each perturbation is calculated by integrating the measured force. Since the sensors themselves are placed on the robot such that they measure force in either the sagittal or coronal planes, the measured impulse is already decomposed into components in the two orthogonal planes.

The input vector used for the estimation was the same as that used for the classification described in Section 4.4, where $\mathbf{x}_k$ was composed of three consecutive joint velocity samples. However, the subset of the data used for training was slightly different, only a single $\mathbf{x}_k$ vector per perturbation was used.

In a similar manner as that for classification, the parameters C, γ and ν for the SVR models were selected using a grid search, with a 10–fold cross validation to prevent over fitting. The metric used for selecting the parameters was based on minimising the mean squared error (MSE) between the true and predicted impulse of the perturbation in each plane.

Upon selection of the best parameters for the two SVR models, the MSE for the magnitude was 0.087 Ns, and the MSE for the direction was 4.2°. Scatter plots of the magnitude and direction are shown in Figure 4.10 and Figure 4.11, respectively. The estimated magnitude and direction are strongly correlated with the true values, having Pearson's correlation coefficients of 0.62 ($p < 0.001$) and 0.99 ($p < 0.001$) respectively.

4.5.2 Discussion of Estimation Results

It is apparent that the two SVR models are much better at predicting the direction of the perturbation, than the magnitude, from the velocity patterns. Given the same models are used to perform both estimations, and that the direction is more important than magnitude, this result is reasonable.

Initially the SVR models were trained on both the orthogonal and omnidirectional perturbation–sets. However, it was noted that the use of the omni–directional perturbation–set during training was unnecessary. A model fitted on only the orthogonal perturbation–set performed just as well as a

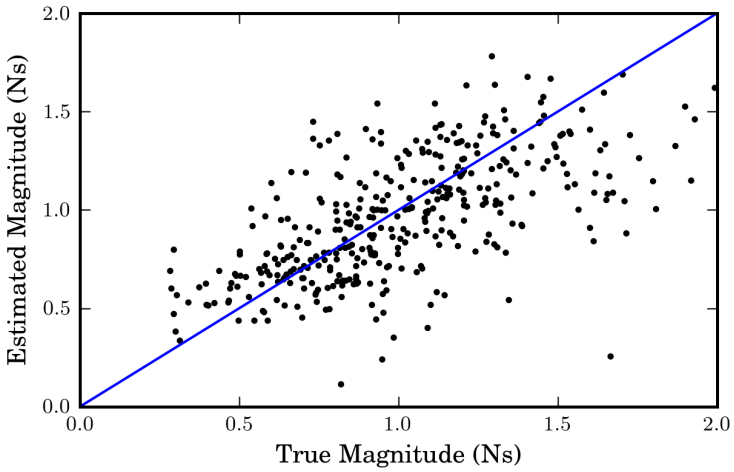


Figure 4.10: A scatter plot of the true and estimated perturbation magnitude.

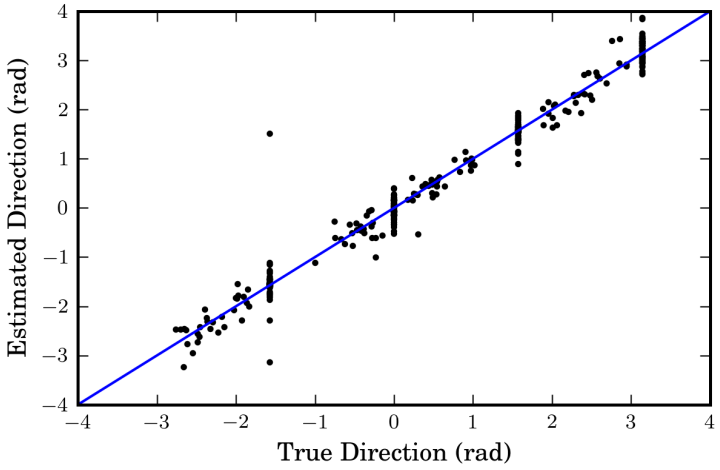


Figure 4.11: A scatter plot of the true and estimated perturbation direction.

model fitted to both datasets. The apparent independence of perturbations in the two planes stems from the configuration of the robot.

The humanoid robot is constructed from servo motors which are only able to rotate around a single axis, and in most cases that axis is either the sagittal or coronal plane. Consequently, a perturbation in the sagittal plane only causes the motors in the sagittal plane to move, and vice versa. This property is not unique to the CycloidII, all humanoid robots have joints in a similar configuration simply because humans also have this property.

A single $\mathbf{x}_k$ vector is used for the calculation of the magnitude and direction, thus the estimation can occur immediately after the perturbation is detected. That is, the mean delay for the perturbation estimation is 138ms.

The estimation of the magnitude and direction of the perturbation can be calculated on the robot in approximately 1.5ms. Combining this with the 1.5ms required to perform the classification, the total time required to execute the system on the robot is 3ms. Given the proprioception refresh rate on the CycloidII is 50Hz, the proposed system uses 15% of the CPU.

4.6 Effect of Low Joint Stiffness on Performance

We saw in Chapter 2 that one of the key properties of human stance is that each joint has quite low stiffness. We previously mentioned in Section 4.2 that to emulate this the robot was given joint stiffnesses as low as possible.

To investigate the importance of low joint stiffness for proprioception based impact perception we collected data at two additional joint stiffness levels; 'medium' and 'high'. The high setting is the default stiffness for the CycloidII and the medium setting was chosen to be midway between the low setting, used elsewhere in this chapter, and the high setting. Only an orthogonal perturbation–set was collected for each of the higher stiffness settings, consequently only the performance of the detection and location stages are considered here.

The procedures described in sections 4.3 and 4.4 were repeated for the two new datasets. A new detector and classifier were trained and optimised for each of the two new settings. The performance of the system with the two new stiffness settings is compared to the low stiffness setting in Table 4.3.

Table 4.3: Effect of Joint Stiffness on System Performance

Stiffness	Avg. Time (ms)	Classification (%)
Low	138	99.8
Medium	190	95.3
High	199	95.6

It is clear that the performance of the system decreases as the joint stiffnesses are increased. The average detection time of a perturbation increases significantly and the location classification success rate reduces. This result suggests that the increased stiffness reduces the movement of the joints, thus reducing the amount of proprioception information available to the system. This reduction in information then limits the performance of the system. Thus, a proprioception based sensing system should use the lowest possible stiffness settings.

The reduction of performance as stiffness is increased is the greatest limitation for the system to be applicable to all humanoid robots. For the system to work, there needs to be considerably movement of the joints upon application of an external force. However, most humanoid robots have adjustable stiffness that can be reduced. For example, the NAO also has adjustable stiffness, early versions of the NAO could not be set with a sufficiently low stiffness, particularly in the arms, but later versions of the NAO were improved to support lower stiffnesses.

4.7 Conclusion

This chapter presents a sensing system capable of detecting, localising and estimating the strength of an external impact to a standing humanoid robot. The proposed system makes use of only proprioceptive information from the robot's servo motors, in particular, the joint velocities. The system was applied to a physical CycloidII humanoid robot where the system performed extremely well, quickly detecting and characterising external forces, as well as being computationally efficient.

The detection of the perturbation is performed through thresholding on a weighted norm of the joint velocities. The weights were optimised to minimise the mean detection time for the upper and lower body, resulting in a detection delay of 138ms. The mean detection delay for perturbations on the upper body was 95ms, while the delay for the lower body was 215ms.

The localisation of the impact was reduced to a classification problem by discretising the possible locations to a set of 16; 8 on the upper body, and 8 on the lower body. A Support Vector Machine was able to classify each impact to the upper body with a perfect 100% accuracy. The addition of a simple voting strategy to the output of the SVM was required to achieve perfect accuracy on the lower body, due to difficulties inherent to the robot hardware.

The direction and strength of the impact were estimated using a pair of orthogonal Support Vector Regression models. After the models were fitted to the data, the mean squared error between the true and predicted values were 0.087Ns for the impulse, and 4.2° for the direction.

Chapter 5

Improvements in Walking through Joint Stiffness Reduction

Contents

5.1 Introduction 70
5.2 Equipment and Method 71
5.2.1 Hardware and Software 71
5.2.2 Optimisation Algorithm and Parameter Space 72
5.2.3 Optimisation Path and Fitness Function 73
5.3 Results 73
5.3.1 Speed 74
5.3.2 Efficiency 74
5.3.3 Stability 76
5.4 Discussion 80
5.5 Conclusion 81

5.1 Introduction

The literature on human locomotion reviewed in Chapter 2 showed that in addition to the the joint trajectories, the joint stiffnesses play a vital role. Recall that the literature showed that the stiffness of each joint is different [75] and that the stiffness varies as a function of gait phase [123, 76, 77].

It then follows that one should optimise both the walk parameters that effect the calculated trajectories and the parameters that effect the stiffnesses. In humanoid robots, the joints are typically controlled by a low–level position control loop; typically a PID controller. It is common for the joint stiffness of humans to be modelled as a PD controller [153]. Thus, the stiffness of each joint on a humanoid robot can be manipulated by modifying the gains of the low–level position controller.

There have been several attempts in the literature to emulate low joint stiffness which typically involve modifications to the high-level controller, for example, an impedance controller [154], or a variable compliance controller [155, 156, 157]. Both of these approaches are implemented with stiff low–level position controllers. However, modelling and sensing errors occur in implementing a compliant controller with hard position tracking. By placing the stiffness control in the low–level controller these errors can be avoided.

In this chapter we explore the advantages of low joint stiffness by adjusting the gains of the low–level position control on a physical NAO robot. We compare a low stiffness walk to two high stiffness walks in terms of walk speed, efficiency and stability. We find that the reduced joint stiffness improves all three areas of the walk.

The optimisation of joint stiffnesses for humanoid robot walking is examined further in Chapters 6 and 8. Chapter 6 examines the advantages of low stiffness with optimised walk parameters from within the context of the meta–optimisation of walk optimisation techniques. Chapter 8 extends the available joint stiffnesses further by allowing the stiffnesses to be gait–phase dependent.

The remainder of this chapter is organised as follows: firstly, the equipment and the method used to select the reduced stiffness values for each joint is described. The observed improvements in the walk speed, efficiency and stability are then presented, followed by a discussion of the results.

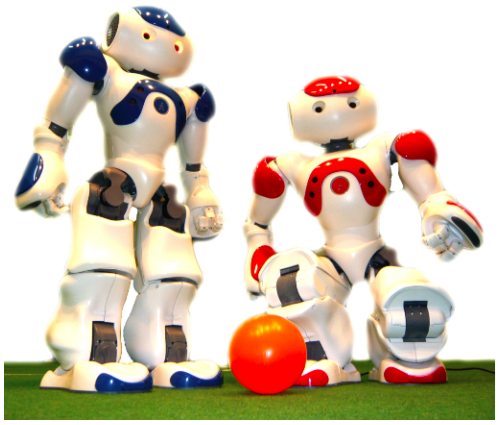


Figure 5.1: The NAO v2 RoboCup edition.

5.2 Equipment and Method

5.2.1 Hardware and Software

The work in this chapter was performed on an early 2008 NAO (v2) humanoid robot, as shown in Figure 5.1, in preparation for RoboCup 2008. At the time, the only walk engine available for the NAO was an alpha version of the Aldebaran walk engine. Being an alpha version the walk engine was quite limited; it had no sensor feedback for walk stabilisation and had few walk parameters. The few available walk parameters could only be adjusted over a limited range and could not be adjusted in real–time. The walk engine was also unable to walk in an omnidirectional, or even continuous, manner.

The NAO uses a PID controller, with very high gains, for the low–level control of the joint positions of the form shown in Figure 5.2. The proportional (K_P), integral (K_I), and derivative (K_D) gains are fixed, however, the parameter K_S can be adjusted. Consequently, K_S can be used to specify the joint stiffness on the NAO. Since K_S is applied after the saturation element it also reduces the maximum torque available to each joint.

Each motor in the NAO is equipped with a current sensor, and the battery has both a voltage and current sensor. These sensors are used to record the current drain in each motor and to calculate the energy used by the robot during a walk trial.

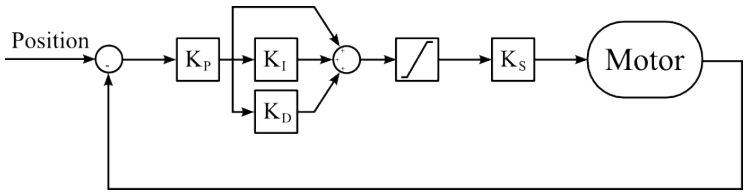


Figure 5.2: A block diagram of the low-level positional control used by the NAO.

5.2.2 Optimisation Algorithm and Parameter Space

Due to the limitations of the NAO and the walk engine provided circa 2008, the implementation of an autonomous optimisation procedure was not possible. Consequently, a simple optimisation algorithm was developed that could be performed manually by a human operator.

To achieve this Powell's method [158] was employed. Essentially, the algorithm adjusts a single walk parameter until a local maxima is reached, after which the algorithm begins adjusting the next parameter. The procedure is repeated until all parameters have simultaneously reached local maxima. The algorithm also uses a fixed relative step size for each parameter which simplifies its manual implementation.

The walk parameter space available for optimisation is shown in Table 5.1. There are 8 traditional walk parameters that effect the calculated joint trajectories. With stiffness included in the optimisation there are an additional 6 stiffness parameters, as the same parameters are used for both the left and right legs.

In this chapter there are three sets of walk parameters discussed. The first set is the default parameters provided by Aldebaran used with a global stiffness of 100%. The second set is the 'high–stiffness' walk which has been selected through optimisation using only the traditional walk parameters. Finally, the 'low–stiffness' walk has been selected through optimisation using a parameter space including both the traditional walk parameters and the additional stiffness parameters.

Table 5.1: Walk Parameter Space for 2008 NAO

Parameter	Min	Max
StepLength (cm)	0	10
StepFrequency (Hz)	1.5	3
StepHeight (cm)	0	3
ZMPX (cm)	-3	3
ZMPY (cm)	-3	3
HipHack (rad)	0	0.3
TorsoPitch (rad)	-0.2	0.4
TorsoHeight (cm)	19	27
HipStiffness	[25,25]	[100,100]
YawStiffness	25	100
KneeStiffness	25	100
AnkleStiffness	[25,25]	[100,100]

5.2.3 Optimisation Path and Fitness Function

Due to the limitations of the walk engine only the forward walk was optimised. Each walk was trialled over a straight path of length 400cm. The robot was allowed to have a moving start and was not required to stop at the end of the path. This allowed the robot to be walking at its maximum speed for the entire length of the path.

The fitness function used for the selection of the best walk parameters was the average speed over the path. The time required to complete the path was measured manually from which the speed was then computed.

5.3 Results

The walk parameters selected by the optimisation process, along with the default settings, are shown in Table 5.2. The table shows that the joint stiffnesses for the low–stiffness walk are significantly different from their default values. Furthermore, upon comparison of the low and high stiffness walk parameters, significant increases in step length and frequency are only possible once the additional stiffness parameters have been introduced.

Table 5.2: Optimised Walk Parameters

Parameter	low–stiffness	high–stiffness	default
StepLength (cm)	5.7	4.5	4.0
StepFrequency (Hz)	2.4	2.2	2.0
StepHeight (cm)	1.2	1.1	1.5
ZMPX (cm)	0	1.0	1.0
ZMPY (cm)	1.0	2.0	1.5
HipHack (rad)	0.03	0.07	0.07
TorsoPitch (rad)	0.02	0.09	0.09
TorsoHeight (cm)	23	19	19
HipStiffness	[30,70]	[100,100]	[100,100]
YawStiffness	60	100	100
KneeStiffness	30	100	100
AnkleStiffness	[30,25]	[100,100]	[100,100]

Table 5.3: Comparison of Walk Performance

Walk	Speed (cm/s)	Cost of Transport (J/Nm)
low–stiffness	13.9	5.8
high–stiffness	11.3	8.6
default	8.7	11.0

5.3.1 Speed

The speeds of the low–stiffness, high–stiffness and default walks are shown in Table 5.3. The speed of the low–stiffness walk was measured to be 13.9±0.2 cm/s. The speeds of the two high stiffness walks were 8.7±0.1cm/s for the default walk, and 11.3±0.2cm/s for the high–stiffness walk. This means that the low–stiffness walk was 60% faster than the Aldebaran walk, and 23% faster than the high–stiffness walk.

5.3.2 Efficiency

A useful quantity to represent the walk efficiency is the specific cost of transport c_{et} [87], given by

$$c_{et} = \frac{E}{mgP}, \tag{5.1}$$

where E is the energy used, m is the mass, and P is the distance travelled.

The cost of transport of each walk is shown in Table 5.3. In terms of cost of transport, the low–stiffness walk represents a 33% and 47% improvement compared to the high–stiffness and default walks, respectively.

Figure 5.3 shows the current drain from the battery while the NAO was walking the length of the optimisation path. The current drawn by the microprocessors was approximately 0.9A and is included in the values shown in the figure. It is clear from the figure that the current consumption of the low–stiffness walk is significantly lower than the two high stiffness walks, implying that the low stiffness walk uses less energy per unit time.

Furthermore, Figure 5.3 shows that the current drain is much smoother for the low–stiffness walk, with only a small difference between peak and average currents. The standard deviations of the unfiltered currents in the figure were 0.14A, 0.27A and 0.25A for the low–stiffness, high–stiffness and default walks, respectively.

The energy used as a function of distance is shown in Figure 5.4. The energy used by the processors (approx. 20W) has been removed from the recorded values for all three walks. The figure demonstrates that the low stiffness walk is also significantly more efficient per unit distance travelled.

A practical result of this improvement in efficiency is the distance the robot can now walk on a single battery charge. The high–stiffness and default walks allow the robot to travel 520m and 415m respectively, while the low–stiffness walk enables the robot to cover 815m, nearly twice the original distance.

The energy consumed by each motor in the left leg is shown in Figures 5.5 and 5.6. The figures show that the energy used as a function of distance is significantly lower in every joint when using low joint stiffness. Furthermore, the energy consumed by each joint is much smoother when the joint stiffness is lowered. This is particularly apparent in Figures 5.5(b) and 5.6(b) where the large spikes in the energy consumption are significantly reduced in the low–stiffness walk.

Table 5.4 summarises the improvement in efficiency of each joint between the low–stiffness and default walks. The reduction of the energy used per centimetre travelled by the CPU is also shown in the table for comparison.

The largest improvements came from the knee, ankle and hip pitch joints, where the efficiencies improved by 68%, 64% and 48% respectively. Of the joints in the leg, the pitch joints move the furtherest while walking forward

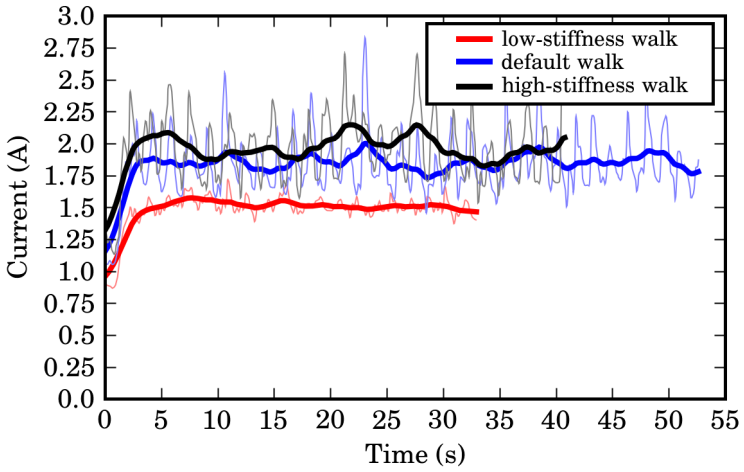


Figure 5.3: The current drain from the battery while walking. The thin lines represent unfiltered current values, and the thick lines represent filtered currents with a moving 2s–window mean. Note that the plots for the two high–stiffness walks are longer because they take more time to cover the optimisation path.

and thus present the greatest opportunity for improvements in efficiency. It is apparent from Table 5.2 that the introduction of low stiffness enabled a significant increase in the height of the centre of mass. This increase is achieved by straightening the legs which also contributes to the reduction of the energy used by the pitch joints.

Table 5.4 also shows which components of the robot use the most energy. The CPU is the greatest consumer of energy, using almost half of the total energy. The knee pitch joint is the next biggest consumer, the primary cause for this is that the robot walks with bent knees. Consequently, the knee requires constant energy to support the robot's weight.

5.3.3 Stability

A heuristic measure of stability was used to compare the low and high stiffness walks. We found the heuristic: the walk is stable if it can complete the straight optimisation path without falling, to be acceptable.

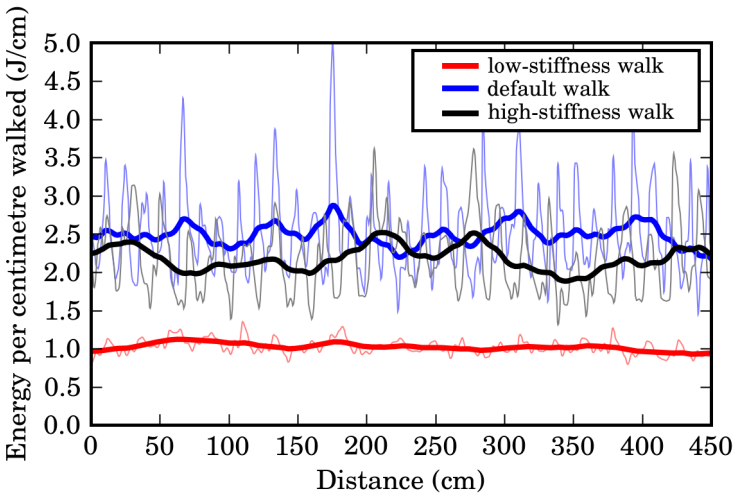
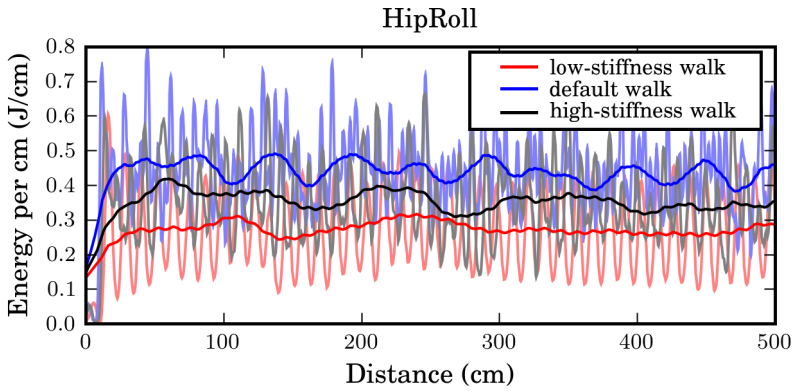


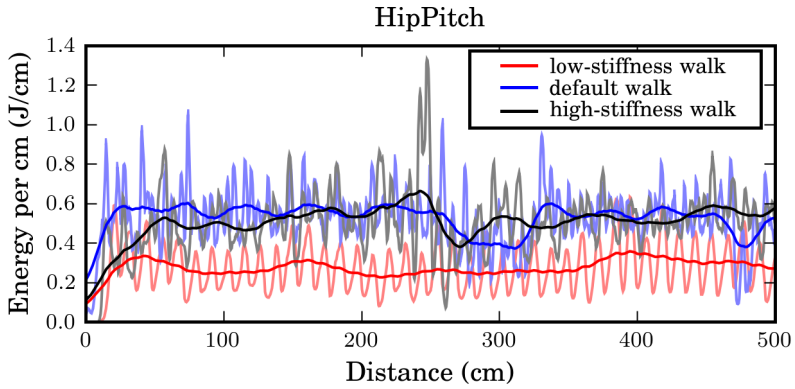
Figure 5.4: The energy per centimetre used by the robot while walking. The thin lines are the unfiltered values, and the thick lines are the moving 2s-window means.

Table 5.4: Improvements in Efficiency of Individual Components

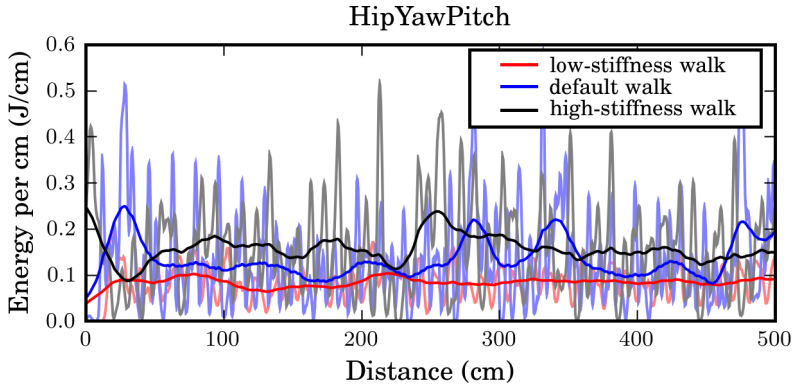
W	Low–stiffness (J/cm)	Default (J/cm)	Improvement (%)
HipRoll	0.28	0.46	38
HipPitch	0.28	0.54	48
HipYawPitch	0.086	0.14	38
KneePitch	0.32	1.0	68
AnkleRoll	0.23	0.34	32
AnklePitch	0.19	0.54	64
CPU	1.4	2.3	38



(a) The hip roll joint.

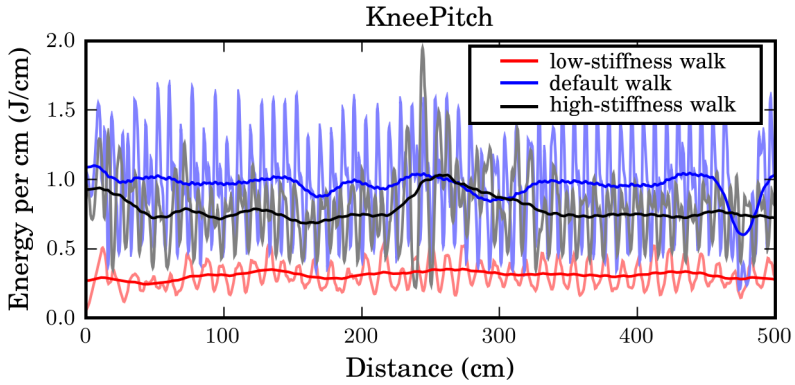


(b) The hip pitch joint.

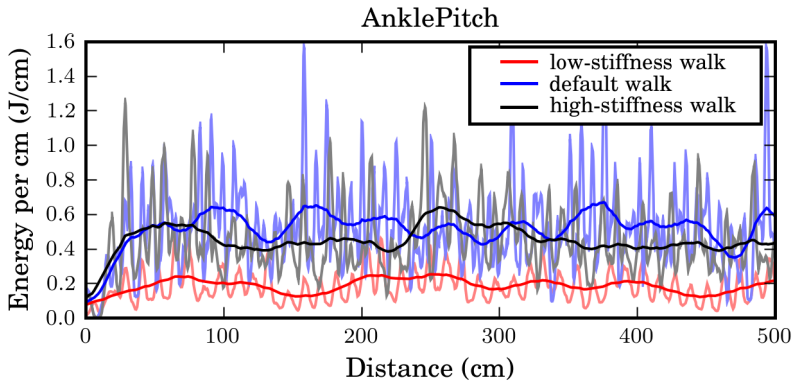


(c) The hip yaw pitch joint.

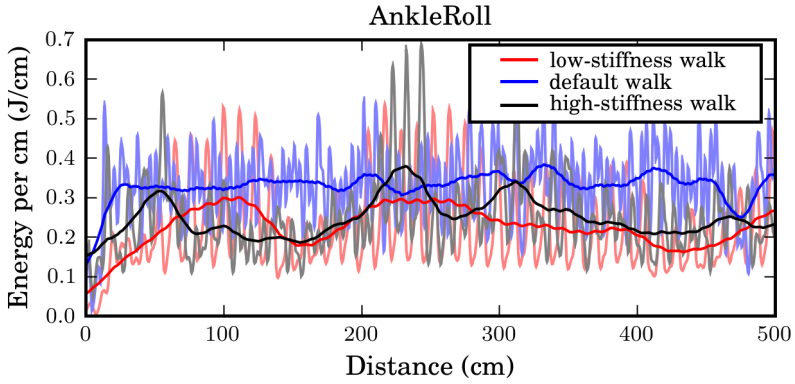
Figure 5.5: The energy used by the upper three joints in the left leg per centimetre travelled while the robot walked along the optimisation path.



(a) The knee pitch joint.



(b) The ankle pitch joint.



(c) The ankle roll joint.

Figure 5.6: The energy used by the lower three joints in the left leg per centimetre travelled while the robot walked along the optimisation path.

Both the default and low–stiffness walks never caused the robot to fall over while traversing the path, post–optimisation, during the collection of speed and efficiency data. This means the robot walked for approximately 100m without falling over. However, the high stiffness walk was not stable. The robot required assistance approximately 30% of the time to prevent it from falling over which was considered acceptable because this particular walk was only constructed for comparative purposes.

The major improvement between the default walk and low–stiffness walk was the robustness to external disturbances. To demonstrate, and quantify, this observation a short length of uneven surface was constructed. The surface was approximately 3m long and had irregularities of amplitude up to 1 cm. The default and low–stiffness walks were each trialled over this surface five times. With the low–stiffness walk the robot successfully completed the path over the irregular surface four out of five times, while with the default walk it completed the course only once in its five trials.

5.4 Discussion

The manufacturer's recommended current drain for the battery was 1.5A. Figure 5.3 shows that the only walk within this recommendation is the low–stiffness walk. Both the default and high stiffness walks are constantly drawing more than the recommended current. Furthermore, Figure 5.3 also shows current peaks of over 2.6A for the two high stiffness walks which is greater than the maximum rated current of the battery. In fact, the robot shut itself down once during testing of the high–stiffness walk to protect the battery.

The inclusion of K_s for each joint as an additional optimisation parameter made tuning the walk easier. It was observed that with a maximum stiffness, it was difficult to find a set of walk parameters that are stable.

One explanation as to why the low stiffness walk is more efficient, is that the low joint stiffnesses prevent the robot from rigidly tracking imperfect trajectories produced by the walk engine. Instead it exploits the natural dynamics of the system, allowing the robot to 'settle' into a more efficient gait, rather than rigidly adhering to the calculated trajectory.

The smoothness of the motion contributes to the improved stability and robustness of the walk. Consider the hip pitch joint during the swing phase.

With stiff tracking the leg is abruptly accelerated forward at the beginning of the swing, the moment exerted on the rest of the body can be great enough to induce slip in the supporting foot, hence rotating the entire robot. This can potentially result in the swing leg hitting the ground, or the robot becoming unbalanced and falling sideways. As the amount of slip is unpredictable this can present a problem. A similar effect was noticed at the end of the swing phase. Reducing the stiffness in the hip pitch joint smoothed out these accelerations and consequently improved the stability of the walk.

The robustness of the walk was also improved by the compliance introduced through the low stiffness in the ankle. When the foot comes into contact with the ground the low stiffness allows the foot to conform to the irregularities in the floor, without effecting the orientation of the rest of the leg. If the robot is not falling over, the leg will be approximately vertical. Thus, when the foot becomes the single support, the support leg will remain approximately vertical, held in place by the controller and the friction in the joint itself. In contrast, stiff tracking would actively rotate the leg, through the ankle, on contact such that it is perpendicular to the ground. This will result in the robot falling over if the ground is not perpendicular to the gravity vector.

5.5 Conclusion

The reduction of the stiffness in the low–level position control vastly improves the walk of the NAO. Improvements in speed, efficiency and stability were observed and quantified. The reduction in stiffness of each joint reduces the rigidity at which the walk pattern trajectory is tracked. This prevents the robot attempting to perfectly track a non-ideal gait, instead allowing it to 'settle' into a more natural and efficient gait. The reduction in stiffness also makes the walk more robust to external influences as the low stiffness in the position tracking prevents the joint from exerting too much torque against an obstacle.

The low stiffness walk was found to be 60% faster than the default walk and 47% more efficient. An uneven surface was used to demonstrate that the low stiffness walk was also significantly more robust. To investigate the effect of the reduction in stiffness alone, an improved high stiffness walk was also created for comparative purposes. Compared to this walk the low–stiffness walk was

23% faster, and 33% more efficient. This demonstrates that the majority of the improvement was due to the reduction in the stiffness.

The improved speed and stability of the low–stiffness walk proved to be advantageous at RoboCup 2008. The low–stiffness walk was significantly faster than the other walks, demonstrated at the competition, whose speeds ranged between that of the default walk and the improved high stiffness walk. The low–stiffness walk presented in this chapter was a major factor in the NUbot's victory at the 2008 RoboCup two-legged standard platform league.

Chapter 6

Meta–optimisation of Walk Optimisation Techniques

Contents

6.1 Introduction 84
6.2 Equipment and Method 85
6.2.1 Hardware and Software 85
6.2.2 Optimisation Path 86
6.2.3 Optimisation Expense 88
6.3 Optimisation Algorithms 89
6.3.1 Evolutionary Hill Climbing with Line Search 90
6.3.2 Policy Gradient Reinforcement Learning 90
6.3.3 Gaussian Particle Swarm Optimisation 93
6.4 Fitness Functions for Optimisation 93
6.4.1 Speed 94
6.4.2 Efficiency 94
6.4.3 Froude–Number 95
6.5 Parameter Spaces for Optimisation 95
6.6 Meta-optimisation of Algorithms 96
6.7 Design of the Comparison of Walk Optimisation Techniques 97
6.8 Comparison of Algorithms 99
6.9 Comparison of Fitness Functions 108
6.10 Comparison of Parameter Spaces 110
6.11 Application to the Physical NAO 112
6.12 Conclusion 115

6.1 Introduction

This chapter presents the design of a walk optimiser through the comparison and meta–optimisation of existing techniques. The design has been performed exclusively in simulation with the final meta–optimised system being applied to a physical robot. At each stage of the design, measures have been implemented to minimise the stress placed on the robot during the optimisation process. Furthermore, priority has been given to maximising the stability of the final optimised walk.

There are three core components of a walk optimiser; the optimisation algorithm, the fitness function and the walk parameter space. In this chapter we examine each component and develop a complete walk optimisation system.

To begin we determine the most suitable optimisation algorithm by comparing three algorithms; Evolutionary Hill Climbing with Line Search (EHCLS) [144], Policy Gradient Reinforcement Learning (PGRL) [159] and Gaussian Particle Swarm Optimisation (GPSO) [147]. Each algorithm was meta–optimised prior to the comparison to both maximise the performance of each algorithm and to enable a fair comparison.

Next we investigate which property of the walk it is best to optimise by comparing three different fitness functions. Fitness functions based on the average speed, efficiency and Froude–number, were evaluated over a predefined path that required omnidirectional walking.

Finally, we compare the performance of two different parameter spaces. The first parameter space consisted of only the traditional walk parameters that effect the joint trajectories calculated by the walk engine. The second parameter space included both the traditional walk parameters and additional parameters to specify the stiffness of each joint in the leg.

The optimisation algorithms, fitness functions and parameter spaces are compared based on the performance of the resultant optimised walks. The speed, efficiency and stability of the walk selected after the robot has experienced a fixed amount of stress during the optimisation process are the primary basis for comparison. The repeatability of the optimisation process with a particular combination of algorithm, fitness function and parameter space is also considered, where a small variance between successive optimisation episodes is preferred.

The remainder of this chapter, firstly describes the equipment used and the optimisation process itself. The meta–optimisation of the existing algorithms is then presented, followed by the comparison of the algorithms, fitness functions and parameter spaces. Finally, the best combination of algorithm, fitness function and parameter space is applied to a physical robot to verify the effectiveness of the proposed optimisation system.

6.2 Equipment and Method

6.2.1 Hardware and Software

The work in this chapter was performed on Aldebaran's NAO robot. This robot was used because it is typical of small humanoid robots and comes equipped with an omnidirectional walk engine. The NAO was also used in the SPL at RoboCup in 2008–2012, thus providing an external application for the optimised walks produced in this chapter.

In particular, a simulated version of Aldebaran's NAO robot, run using the Webots software package [139] was used for the meta–optimisation and comparison. The model of the NAO provided with the simulation software was modified to more accurately reflect the masses of the limbs of the physical robot. The simulation model was supplemented with additional sensors to measure the energy consumed while walking, including a simulation of the energy used by the internal CPU of the NAO.

The physical robot used to validate the optimisation system designed in simulation was a 2010 NAO (v3.2). The two humanoid robot platforms are shown in Figure 6.1.

The NUPlatform software framework discussed in Chapter 3 was used on both platforms. The NUPlatform software framework had been developed to allow the same software to be executed on different robot platforms. This allows identical walk optimisation techniques to be evaluated across multiple robot platforms without the need to modify or rewrite the software.

The proprietary walk engine of Aldebaran [61] was used for the physical NAO. As this walk engine is closed–source and limited to running only within Aldebaran's own middleware, an alternative walk engine was required for the simulator. An open–source walk engine based on a Zero Moment Point preview

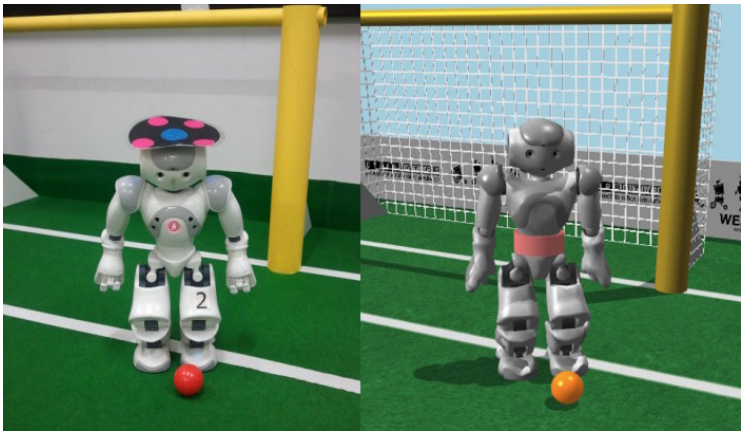


Figure 6.1: The physical and simulated NAO robot. The physical NAO (v3.2) is pictured with an overhead ssl–vision marker.

controller [160] was selected for both its similarity to the proprietary engine and its availability [135]. The walk parameters and their corresponding ranges, for the two walk engines, are shown in Table 6.1.

The initial walk parameters for the simulated NAO were selected manually. This was necessary for the simulated NAO because there was no existing set of stable parameters suitable for the platform. The default walk parameters for the physical NAO were sufficient to be used as a starting point for optimisation.

6.2.2 Optimisation Path

It is common to only optimise the forward walk of a humanoid robot [64, 59, 60, 161]. However, here we are concerned with the optimisation of an omnidirectional walk.

One approach to the optimisation of an omnidirectional walk engine is to independently optimise a discrete set of walk directions and then use interpolation [62]. This approach significantly increases the number of trials required to perform the walk optimisation. Furthermore, smooth interpolation between different parameter sets is not always possible, for example, interpolating between different centre of mass heights and step frequencies is difficult.

An alternative approach is to use a single set of parameters for every walk direction. A circular path is used in [162] to optimise an omnidirectional walk

Table 6.1: Walk Parameters

	Simulator		Physical	
Parameter	Min	Max	Min	Max
Velocities	[0,0,0]	[70,70,2]	[0,0,0]	[30,30,2]
Accelerations	[0,0,0]	[140,140,4]	[0,0,0]	[140,140,4]
StepFrequency	1 Hz	5 Hz	1.6 Hz	4.5 Hz
StepHeight	0 cm	8 cm	0.5 cm	6 cm
ZMPStatic	0	1	-	-
ZMPOffset	0 cm	7 cm	-	-
SensorGain	[0,0]	[0.2,0.2]	-	-
SensorSpring	[0,0]	[1000,1000]	-	-
DSFraction	0.1	0.8	-	-
HipHack	0 rad	0.3 rad	0 rad	0.2 rad
FootLift	-0.4 rad	0.4 rad	-	-
TorsoPitch	-0.2 rad	0.4 rad	-0.15 rad	0.15 rad
TorsoHeight	25 cm	32 cm	27 cm	32 cm
HipStiffness	[30,30]	[100,100]	[30,30]	[100,100]
YawStiffness	30	100	30	100
KneeStiffness	30	100	30	100
AnkleStiffness	[30,30]	[100,100]	[30,30]	[100,100]

engine. However, this path fails to encompass all of the possible walk primitives, for example, sideward walking. A much more elaborate technique is employed by [163], where an omnidirectional walk is optimised over many different paths. This optimisation was performed in simulation, and consequently, little regard was given to the time required to perform the optimisation and the stress placed on the robot.

For the research conducted here, the fitness of a set of parameters was evaluated using the path shown in Figure 6.2. The path was designed to include forward, sideward and diagonal movements, as well as an omnidirectional backward spin. Additionally, the robot is required to stop and start at the endpoints, further testing the agility of the walk.

To follow the path, a localisation system is required. In the simulator an artificial GPS and compass sensor are used to provide accurate information to control the robots movement. On the physical robot the overhead ssl–vision tracking system [164] is used to provide the same information. The robot's vision system is not used as it is inaccurate and consumes most of the available processing power.

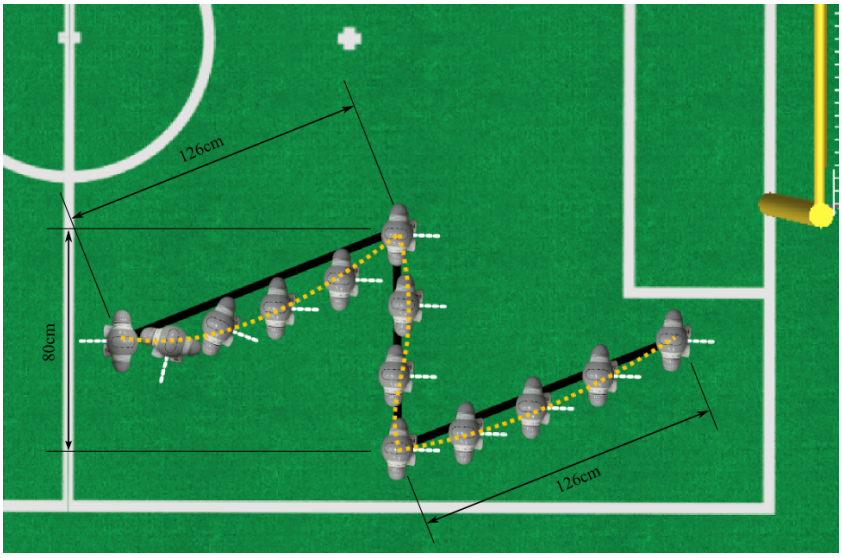


Figure 6.2: The path over which the fitness of a set of walk parameters is determined. The black line indicates the desired path, the dotted line displays a typical path travelled by a robot, and the dashed line emanating from the robot indicates its orientation.

During optimisation a walk trial is ended when either the last waypoint is reached, or the robot falls. If the last waypoint is reached, the next trial is started immediately with the waypoints mirrored. This allows the optimisation process to run continuously without the need to reset the robot to a particular starting position. If the robot falls, a get–up routine is executed, and the robot returns to the starting point to begin the next trial.

6.2.3 Optimisation Expense

To reflect the stress placed on the robot during the optimisation process we introduce an expense measure. The expense of the optimisation process, $\mathcal{E}$, after i walk trials and f falls, is given by

$$\mathcal{E} = i + 9f. \tag{6.1}$$

The 1:9 ratio between successful and unsuccessful iterations was selected based on the relative maintenance costs of the wear associated with completing the path and with falling.

We use the expense in place of the iteration count and a fixed expense of $\mathcal{E} = 2000$ is used as a stopping criterion for the optimisation process. By using this measure of expense in the meta–optimisation of parameters for optimisation algorithms, we select a combination that reduces the stress placed on the robot during the optimisation process. Effectively, the 1:9 expense ratio penalises the optimisation 9 times more for a trial where the robot falls as compared to a successful trial.

The expense measure is also used throughout the comparison of optimisation techniques. Using the expense allows for an effective comparison between techniques that have a significantly different number of unsuccessful trials.

6.3 Optimisation Algorithms

There are numerous algorithms for walk optimisation [66, 67], of which some are more suitable for application to humanoid robots. The algorithms in the literature can be separated into three categories; local, hybrid and global optimisers.

A local optimiser starts from an initial set of parameters and iteratively moves to a local extremum. For walk optimisation, the requirement of an initial starting point translates to finding an initially stable walk. The preference for an initially stable walk improves convergence and reduces the stress on the robot, especially in the early stages of optimisation. However, finding such a parameter set is nontrivial. Another problem with local optimisers is that they only converge to a local extremum, which may be far from the global optimum.

A hybrid optimiser is a local optimiser that has been modified to escape from local extrema. Typically, the modification involves a reset phase where the optimisation is either restarted from a different point, or the algorithm backtracks to a previous point and searches in a new direction.

A global optimiser seeks a global extremum through a much wider search of the parameter space compared to both the local and hybrid optimisers. Typically, a global optimiser does not require a starting point, or is not highly dependent on a good starting point. This is beneficial for walk optimisation

because one does not need to manually search for a good starting walk. However, compared to local optimisers, global optimisers have significantly slower convergence speed.

Three algorithms were chosen from the literature to represent each category; PGRL, a purely local optimiser; EHCLS, a hybrid optimiser; and GPSO, a global optimiser.

The PGRL algorithm is a gradient ascent approach, where the gradient is estimated from a small set of experiments at each iteration. The PGRL algorithm was chosen to represent the local optimisation class because it has been shown to perform quite well in the domain of walk optimisation [159, 68, 59, 162].

The EHCLS algorithm, as the name suggests, is a simple hill climbing algorithm with an additional gradient ascent component. The algorithm also includes a reset phase to help escape from local extrema, making the algorithm an example of a hybrid optimiser. This algorithm has also been shown to perform well in the walk optimisation domain [165, 144, 66].

The GPSO algorithm is an example of a global optimiser and was chosen for its high convergence speed and simplicity. The algorithm does not need a starting point, thus avoiding the initial manual search for a stable set of walk parameters that could bias the final optimised walk.

6.3.1 Evolutionary Hill Climbing with Line Search

A modified version of the EHCLS algorithm proposed in [144] was used. The calculation of the size of the mutation in each dimension was modified to explicitly use the limits of the parameter space in that dimension. With this modification, the size of the mutation is specified intuitively and does not require meta–optimisation. The pseudo–code for the algorithm is shown in Algorithm 1.

6.3.2 Policy Gradient Reinforcement Learning

A modified version of the PGRL algorithm proposed in [159] was used. The algorithm includes a modified calculation of the adjustment vector A. The vector is compressed using a sigmoid function in place of the normalisation in the original algorithm. Primarily, this modification improves the ability of the

Algorithm 1 EHCLS

$\theta_{best} = \theta = InitialParameters$
loop
  $\mathcal{F} = \texttt{calculateFitness}(\theta)$
  if $\mathcal{F} > \mathcal{F}_{best}$ **then**
    $\theta_{previousbest} = \theta_{best}$
    $\theta_{best} = \theta$
    $I_{curr} = \mathcal{F} - \mathcal{F}_{best}$
    $\alpha = 0.95 \tanh\left(\frac{I_{curr}}{I_{prev}}\right)$
  end if
  if $StallCount > L$ **then**
    $\mathcal{F}_{best} = \rho\mathcal{F}_{best}$
    $\alpha = 0$
  end if
  $\theta = \theta_{best} + \alpha(\theta_{best} - \theta_{previousbest}) + (1-\alpha)\cdot\mathcal{N}(0, \eta \exp\left(\frac{StallCount}{L} - 1\right))\cdot\texttt{range}(\theta)$
end loop

---

where

θ = current set of walk parameters
$\theta_{best}, \theta_{previousbest}$ = current and previous best sets of walk parameters
$\mathcal{I}_{curr}, \mathcal{I}_{prev}$ = current and previous improvement in best fitness
$\mathcal{F}, \mathcal{F}_{best}$ = current and best fitness of walk parameters
$\texttt{calculateFitness}(\theta)$ = walk fitness evaluation function
$\mathcal{N}(a, b)$ = normal distribution with mean a and variance b
$\texttt{range}(\theta) = \{\theta_{max} - \theta_{min}\}_1^D$ the size of the parameter space for each dimension D

and the following are parameters to be meta–optimised

η = size of the mutation at each iteration
L = limit at which a stall is detected
ρ = amount to reset after a stall.

algorithm to handle large differences in fitness between policies. In particular, policies that have a near zero fitness which occur frequently when a policy results in the robot falling. The pseudo–code for the algorithm can be found in Algorithm 2.

Algorithm 2 PGRL

$\theta = InitialParameters$
loop
  $\{\theta\}_1^N = \texttt{generatePolicies}(\theta, \epsilon)$
  $\mathcal{F} = \texttt{calculateFitness}(\{\theta\}_1^N)$
  for $i = 0$ to D **do**
    $Avg^i_{+\epsilon} \leftarrow$ average fitness for all θ^k that have a positive perturbation in dimension i
    $Avg^i_0 \leftarrow$ average fitness for all θ^k that have no perturbation in dimension i
    $Avg^i_{-\epsilon} \leftarrow$ average fitness for all θ^k that have a negative perturbation in dimension i
    if $Avg^i_0 > Avg^i_{+\epsilon}$ and $Avg^i_0 > Avg^i_{-\epsilon}$ **then**
      $A_i = 0$
    else
      $A_i = Avg^i_{+\epsilon} - Avg^i_{-\epsilon}$
    end if
  end for
  $A = \eta \tanh(A) \cdot \texttt{range}(\theta)$
  $\theta = \theta + A$
end loop

where

$\{\theta\}_1^N$ = set of N walk parameter sets
D = dimension of θ
`generatePolicies(a,b)` = generates N walk parameter sets such that $\theta^k = \{\theta_1 + \delta_1^k, \ldots, \theta_D + \delta_D^k\}$ where each δ_i^k is chosen randomly from $\{-\epsilon_i, 0, \epsilon_i\}$ where $\epsilon_i = \epsilon \cdot \texttt{range}(\theta_i)$
A = adjustment vector $\texttt{range}(\theta) = \{\theta_{max} - \theta_{min}\}_1^D$ the size of the parameter space for each dimension D

and the following are parameters to be meta–optimised

η = specified the step size
ϵ = size of the perturbation applied to each dimension
N = number of policies

6.3.3 Gaussian Particle Swarm Optimisation

The GPSO algorithm in [147] was used with an initial swarm randomly distributed over the entire search space. The particle values are restricted such that $\theta_i^{min} \leq \theta_i \leq \theta_i^{max}$ for all i, and the particle velocities are restricted such that $|\dot{\theta}_i| \leq 0.5(\theta_i^{max} - \theta_i^{min})$. The pseudo–code for the algorithm is shown in Algorithm 3.

Algorithm 3 GPSO

$\{\theta\}_1^N = \texttt{generateParticles}(N)$
loop
  $\mathcal{F} = \texttt{calculateFitness}(\{\theta\}_1^N)$
  $\{p_{best}\}_1^N \leftarrow$ personal best for each particle
  $g_{best} \leftarrow$ global best of all particles
  $\{\dot{\theta}\}_1^N = |\mathcal{N}(0,1)|(\{p_{best}\}_1^N - \{\theta\}_1^N) + |\mathcal{N}(0,1)|(g_{best} - \{\theta\}_1^N)$
  $\{\theta\}_1^N = \{\theta\}_1^N + \{\dot{\theta}\}_1^N$
end loop

where

$\texttt{generateParticles}(N)$ = generates N randomly distributed particles $\mathcal{N}(0,1)$ = normal distribution with mean 0 and variance 1

and the following parameter is to be meta–optimised

N = number of particles

6.4 Fitness Functions for Optimisation

There are many aspects that can be considered in determining the fitness of a gait. The three core components of most fitness functions are the speed, efficiency and stability of a set of parameters.

An argument can be made for optimising the walk speed of a robot in a competitive environment, as often the first robot to complete the task is the winner. RoboCup's humanoid soccer leagues are examples of such an environment, where being the first robot to the ball offers a significant advantage. Furthermore, in other applications improving the walk speed can maximise the amount of work performed by the robot in a given amount of time.

The efficiency of a walk is also an important property as it dictates the distance the robot can travel on a single battery charge. There are other advantages to an improved efficiency including lower operating temperatures and lower energy costs.

Here we propose three fitness functions based on the speed, the efficiency and the Froude–number. The stability of a set of parameters is measured implicitly in each fitness function by the ability to complete the relatively long optimisation path described in Section 6.2.2.

6.4.1 Speed

The speed–based fitness function, $\mathcal{F}_S$, is given by:

$$\mathcal{F}_S = \begin{cases} \frac{D}{P} \cdot \frac{0.5D}{T} & \text{if path not completed} \\ \frac{P}{T} & \text{if path completed} \end{cases} \tag{6.2}$$

where D is the distance travelled, P is the distance of the desired path shown in Figure 6.2 and T is the time required to complete the path.

For a set of walk parameters that successfully complete the path, the fitness is simply the average speed. The path distance is used instead of the actual distance travelled to avoid rewarding the robot for deviating from the path. However, if the path is not completed, the fitness is reduced by a half, to penalise the parameter set for being unstable. It is also scaled by the ratio of the distance travelled and the complete path distance, to penalise parameters that cause the robot to fall quickly.

6.4.2 Efficiency

The cost of transport [87], c_{et}, is given by:

$$c_{et} = \begin{cases} \frac{P}{D} \cdot \frac{E}{0.5mgP} \cdot & \text{if path not completed} \\ \frac{E}{mgP} & \text{if path completed} \end{cases} \tag{6.3}$$

where E is the total amount of energy consumed completing the path and m is the mass of the robot. The energy is calculated using voltage and current measurements from the battery and motors.

The c_{et} is then used to calculate the efficiency–based fitness function, $\mathcal{F}_C$, using:

$$\mathcal{F}_C = \frac{180}{4 + c_{et}}. \tag{6.4}$$

Note that this fitness function was designed to give $\mathcal{F}_C$ similar numerical properties to $\mathcal{F}_S$, that is, it has a comparable range of values and is a maximum when the efficiency is the highest. This scaling allows the fitness functions to be interchanged without modifications to the optimisation algorithms.

6.4.3 Froude–Number

The fitness function based on the Froude–number for bipedal walking [166] is given by:

$$\mathcal{F}_F = 20\frac{\mathcal{F}_S^2}{g\ell} \tag{6.5}$$

where ℓ is the leg length of the robot and is measured kinematically for each set of parameters. The scaling factor of 20 is used to give this function a similar range to that of $\mathcal{F}_S$ and $\mathcal{F}_C$.

The primary difference between $\mathcal{F}_F$ and $\mathcal{F}_S$ is the introduction of the leg length. The effective leg length is adjusted by bending the knee, and can vary over quite a large range for the humanoid robots considered here.

6.5 Parameter Spaces for Optimisation

It was shown Chapter 5 by a preliminary investigation that the reduction of joint stiffness improves the walk in every aspect. The same additional joint stiffness parameters are used in this chapter where their effect is more rigorously analysed through simulation.

Formally, the traditional walk parameters are the set

$$\mathcal{W}_p = \{p_1, p_2, \ldots, p_D\}$$

where D is the number of walk parameters for a particular walk engine, either 11 or 19 for the two walk engines considered in this chapter.

The stiffness parameters, s_i, for a single joint specify the percentage of the maximum controller gain,

$$\mathcal{W}_s = \{s_1, s_2, \ldots, s_M\}$$

where M is the number of joints. We adjust the joint stiffnesses such that the values for the left and right legs are the same, thus $M = 6$ for each humanoid robot considered here.

The two parameter spaces considered here are then the traditional walk parameter space, $\mathcal{W}_p$, and the combined traditional and stiffness parameter space, $\mathcal{W}_p \cup \mathcal{W}_s$.

We should note that in Chapter 5 a v2 NAO was used whereas in this chapter a v3.2 NAO was used. There were many improvements to the NAO between these two versions, however, the PID controller shown in Figure 5.2 was used in both versions. Thus, the joint stiffness can be adjusted through the manipulation of K_s.

The simulated NAO uses a proportional controller for low–level joint position control whose K_P can be adjusted at every simulation step. Thus, the joint stiffness is adjusted via the proportional gain in simulation.

6.6 Meta-optimisation of Algorithms

To adequately compare the optimisation algorithms, each needs to be meta–optimised. For the purposes of meta–optimisation the algorithms attempted to improve the speed based fitness function, $\mathcal{F}_S$, with the parameter space including the additional stiffnesses parameters ($\mathcal{W}_p \cup \mathcal{W}_s$).

Given the small number of algorithm parameters requiring meta–optimisation, a grid search was used. Four to six instances of the simulation at each grid point were run in parallel to improve the certainty of the measurement. The fitness function used to evaluate the performance of the algorithm parameters during the meta–optimisation was the mean speed of the final optimised gaits, over the four to six instances, after a fixed expenditure of $\mathcal{E} = 2000$.

Table 6.2: Parameters Selected by the Meta–Optimisation

(a) EHCLS

Parameter	Value
η	0.1
L	5
ρ	0.995

(b) PGRL

Parameter	Value
η	0.05
ϵ	0.05
N	12

(c) GPSO

Parameter	Value
N	30

Recall, η is the step size and N is the number of policies/particles. L and ρ are the reset limit and reset fraction for the EHCLS, respectively, and ϵ is the perturbation size for PGRL.

The parameters selected for each of the three algorithms can be found in Table 6.2. The reader is referred to Section 6.3 where algorithms 1, 2 and 3 describe the relevant tuning parameters for each of the three algorithms.

The EHCLS algorithm performed best with a low L and a high ρ, indicating that the algorithm quickly resets and discards the gradient component when it fails to result in further improvements.

The PGRL algorithm performed best when the two parameters η and ϵ have similar values. This prevents the algorithm from overstepping the area under which its gradient estimate is useful. The selection of N is a trade–off between increasing the convergence speed and improving the quality of the estimated gradient.

The GPSO algorithm has only a single parameter that can be meta–optimised reducing the grid search to a simple one–dimensional interval search. The algorithm performed best with a swarm size of $N = 30$, slightly larger than the number of dimensions.

6.7 Design of the Comparison of Walk Optimisation Techniques

To effectively compare the algorithms, fitness functions and parameter spaces a complete factorial design was used to evaluate each component over all combinations. A graph representing the design of the comparison is shown in Figure 6.3. Essentially, each optimisation algorithm is tested with every combination of fitness function and parameter space. Similarly, each fitness function is tested with every combination of algorithm and parameter space; and each

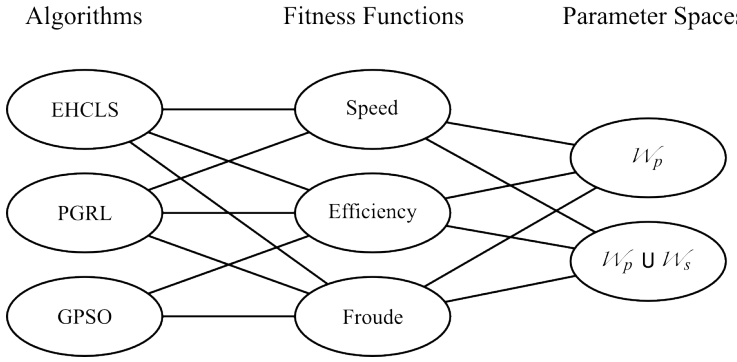
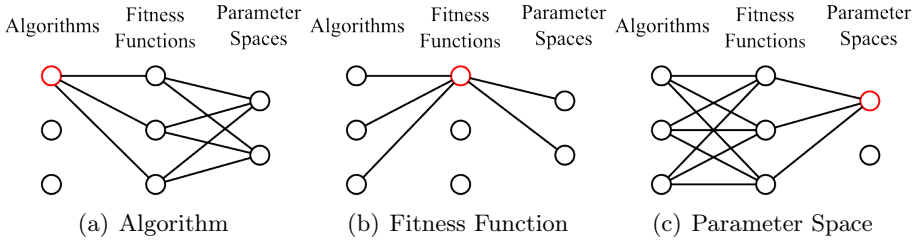


Figure 6.3: The complete factorial design of the comparison of algorithms, fitness functions and parameter spaces.



(a) Algorithm (b) Fitness Function (c) Parameter Space

Figure 6.4: Example graphs illustrating how the complete factorial design relates to the individual comparison of algorithms, fitness functions and parameter spaces. The combinations relevant to the first algorithm, fitness function and parameter space are shown.

parameter space is tested with every combination of algorithm and fitness function. There are a total of 18 combinations of algorithms, fitness functions and parameter spaces in the comparison.

Figure 6.4 further illustrates the design through an example for each component. For a particular algorithm there are 6 combinations of fitness functions and parameter spaces. The performance of an individual algorithm is then the median of the performances of each combination. This experimental design offers the advantage of providing both an insight into the specific performance of the algorithms with each combination and their performance in general. Similarly, there are 6 and 9 combinations for each fitness function and parameter space, respectively.

As each algorithm has a stochastic component and the fact that noise is added to the sensors in the simulator, there is a variation in performance between experiments with the same combination of algorithm, fitness function and parameter space. To assess the variance of each combination, 12 simulation trials were conducted for each combination. The performance of an individual combination is then the median over the 12 simulation trials. Recall, that there were 18 combinations of algorithms, fitness functions and parameter spaces, thus a total of 216 simulation trials were conducted to compare each component of the walk optimiser.

A summary of the results from the 216 simulation trials can be found in Figures 6.5–6.10. Each figure illustrates the median speed and efficiency for a single algorithm–fitness–space combination as a function of expense. The median is used due to its robustness to outliers. The variation between trials for each combination is also shown in the figures. The shaded region indicates this variation with the 25th and 75th percentiles.

Sections 6.8, 6.9 and 6.10 draw results from this complete factorial design. Essentially, the median performance from the relevant combinations in Figures 6.5–6.10 is used to compare the algorithms, fitness functions and parameter spaces directly.

6.8 Comparison of Algorithms

Figures 6.5–6.10 show that after meta–optimisation all of the algorithms perform well with each combination of fitness function and parameter space; improving the walk speed and efficiency significantly compared to the initial walk parameters. We note that this is the case even though the meta–optimisation of algorithm parameters was performed using only $\mathcal{F}_S$ and the parameter space $\mathcal{W}_p \cup \mathcal{W}_s$.

Figure 6.11 and Table 6.3 compare the three algorithms directly. Figure 6.11 shows the evolution of the median speed and efficiency of all of the combinations shown in figures 6.5–6.10 for each of the three algorithms. The figure shows that the GPSO algorithm has a significantly slower convergence in the early stages of the optimisation, while the EHCLS and PGRL are quite similar throughout.

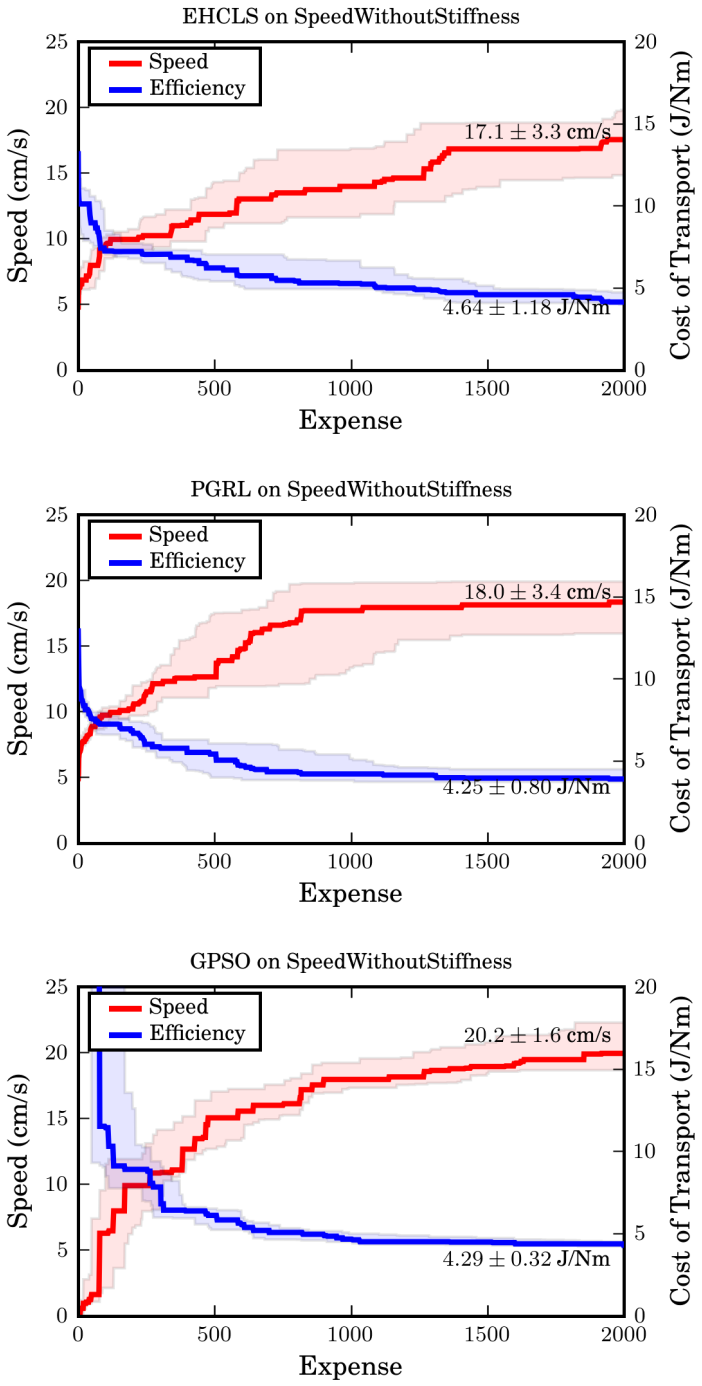


Figure 6.5: The evolution of walk performance for each algorithm when $\mathcal{F}_S$ is used without the additional stiffness parameters ($\mathcal{W}_p$). The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

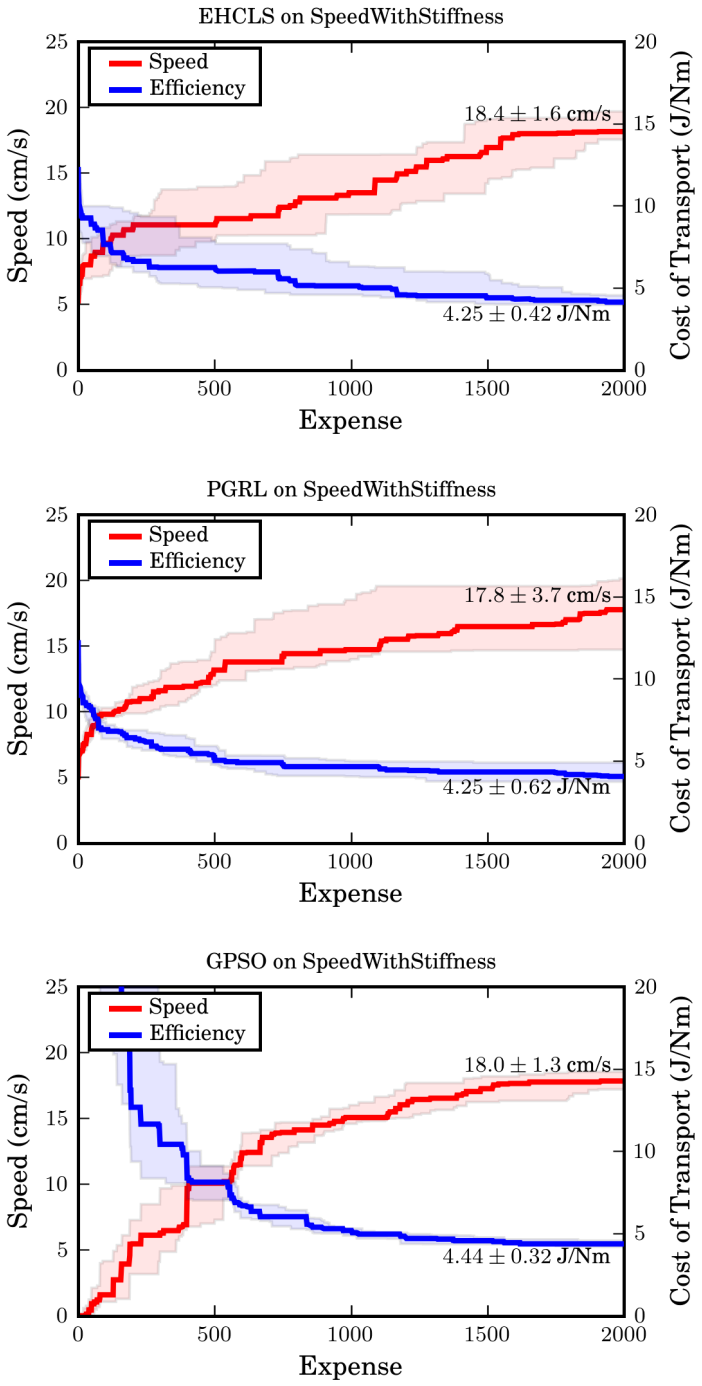


Figure 6.6: The evolution of walk performance for each algorithm when $\mathcal{F}_S$ is used with the additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$). The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

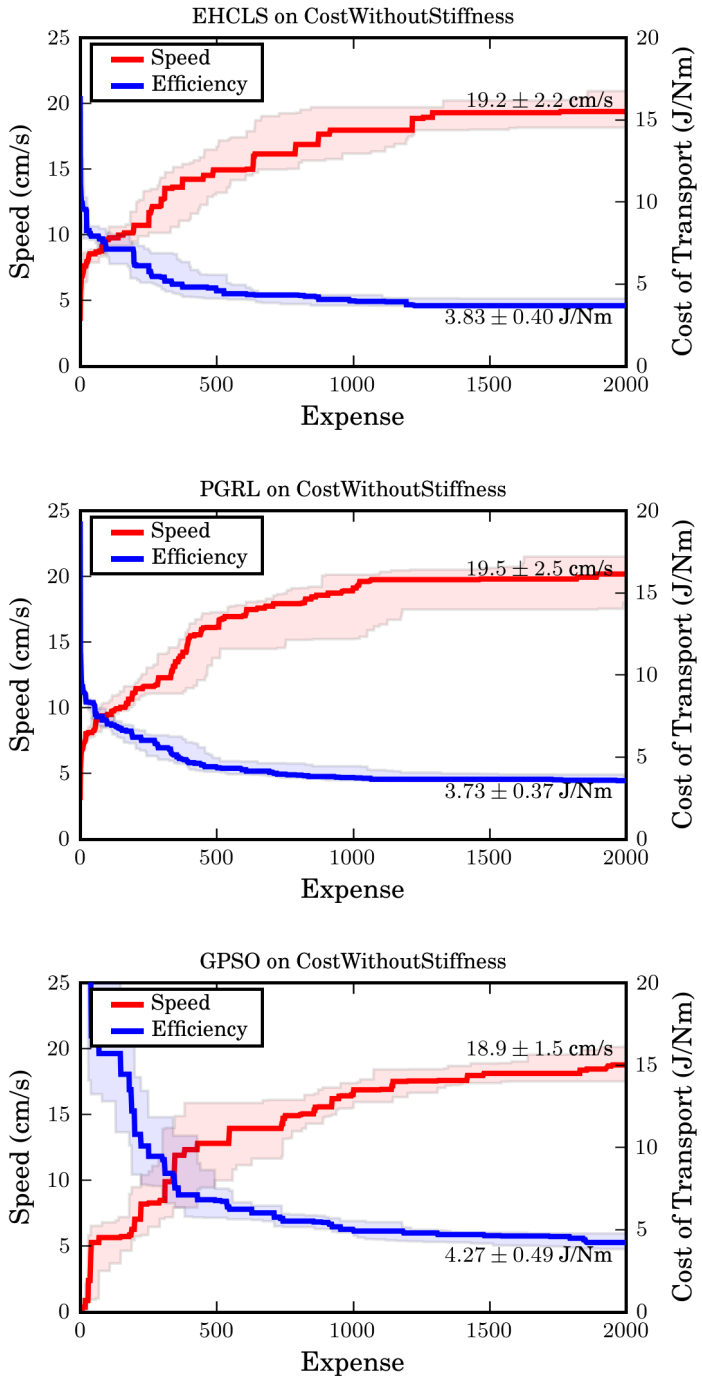


Figure 6.7: The evolution of walk performance for each algorithm when $\mathcal{F}_C$ is used without the additional stiffness parameters ($\mathcal{W}_p$).The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

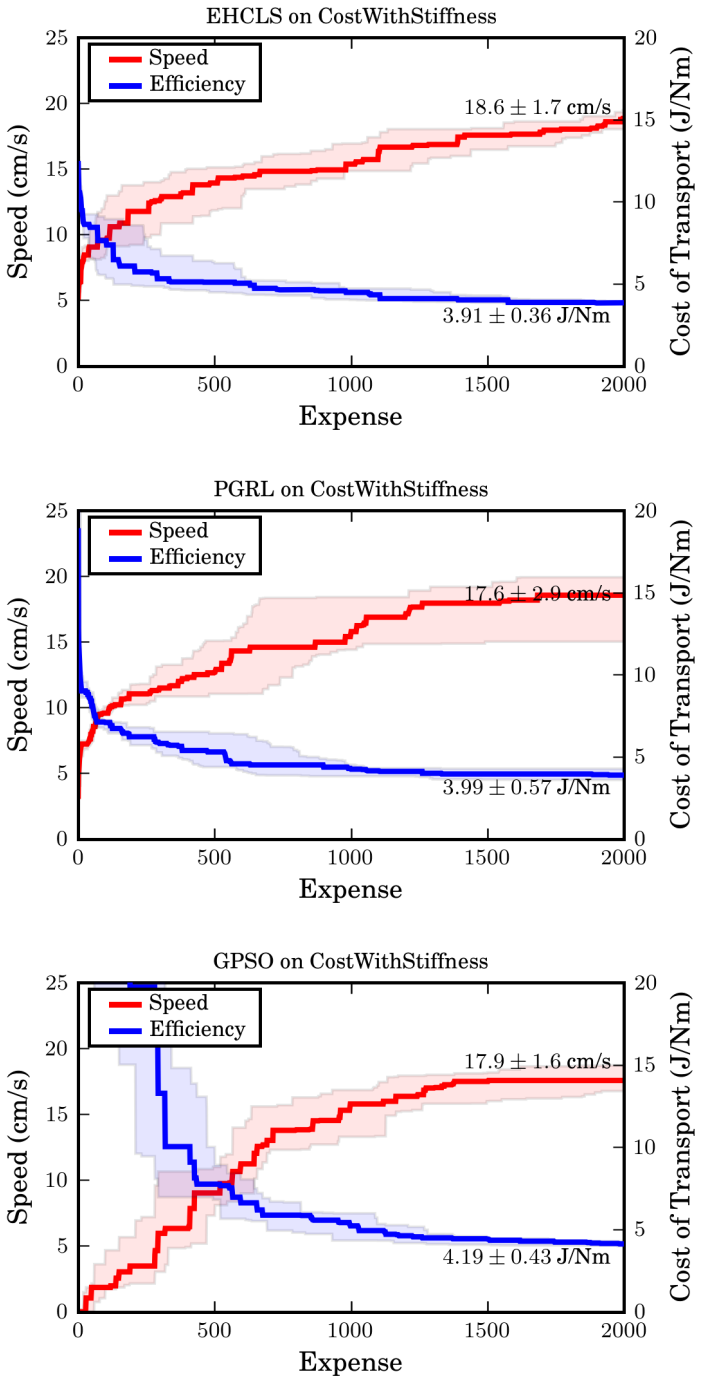


Figure 6.8: The evolution of walk performance for each algorithm when $\mathcal{F}_C$ is used with the additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$). The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

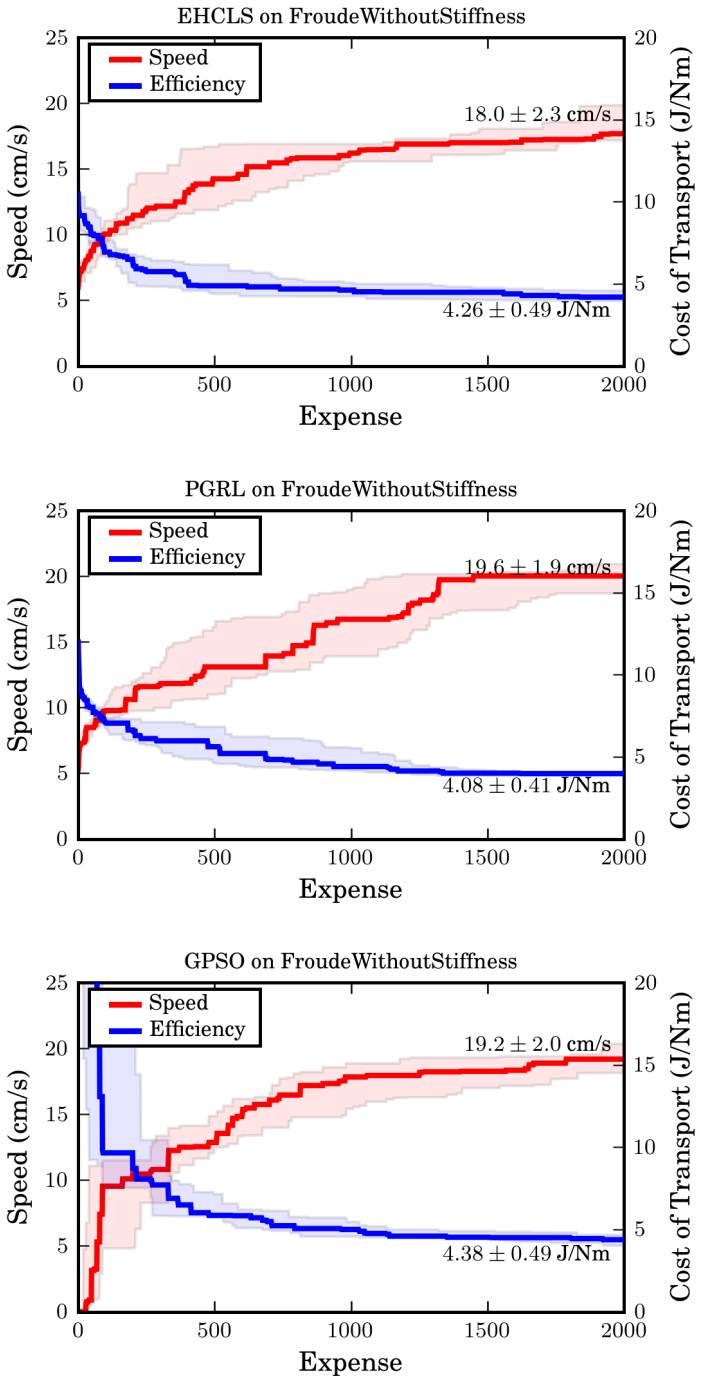


Figure 6.9: The evolution of walk performance for each algorithm when $\mathcal{F}_F$ is used without the additional stiffness parameters ($\mathcal{W}_p$). The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

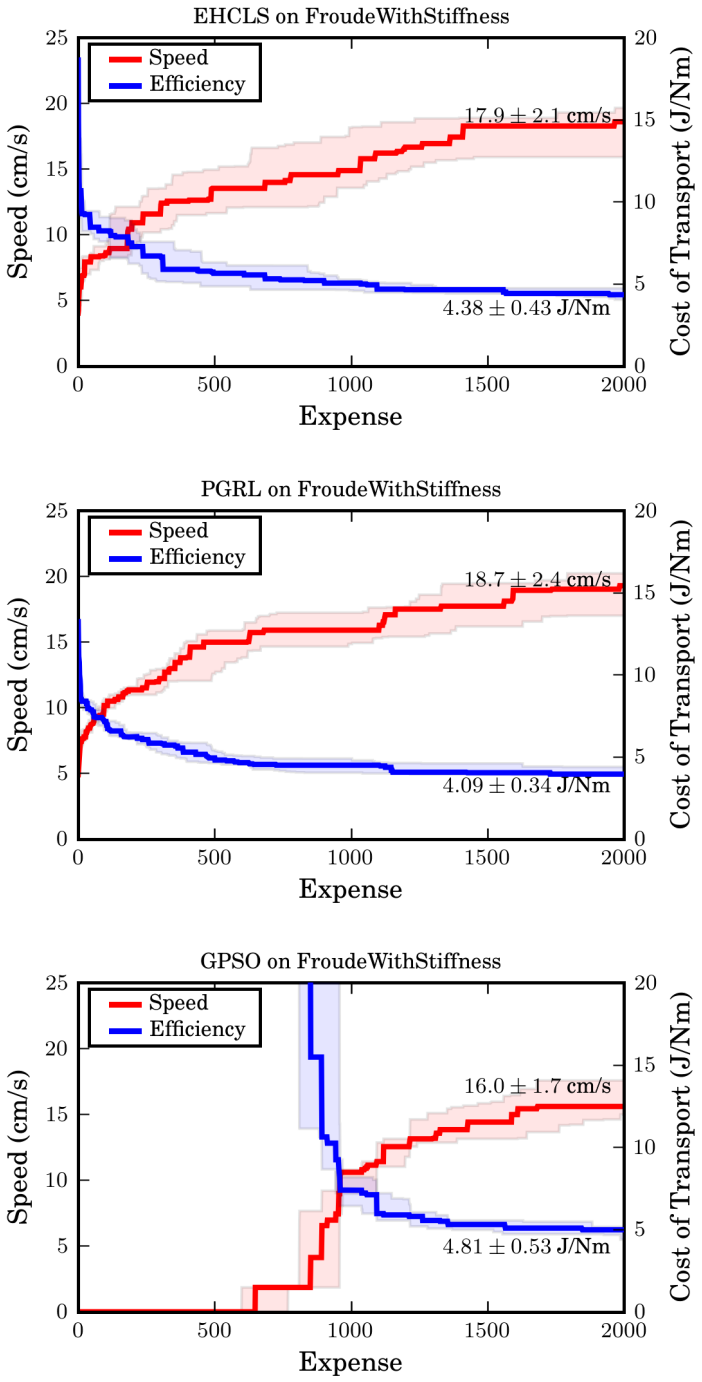


Figure 6.10: The evolution of walk performance for each algorithm when $\mathcal{F}_F$ is used with the additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$). The median speed and efficiency are shown, with the shaded regions bounded by the 25th and 75th percentiles.

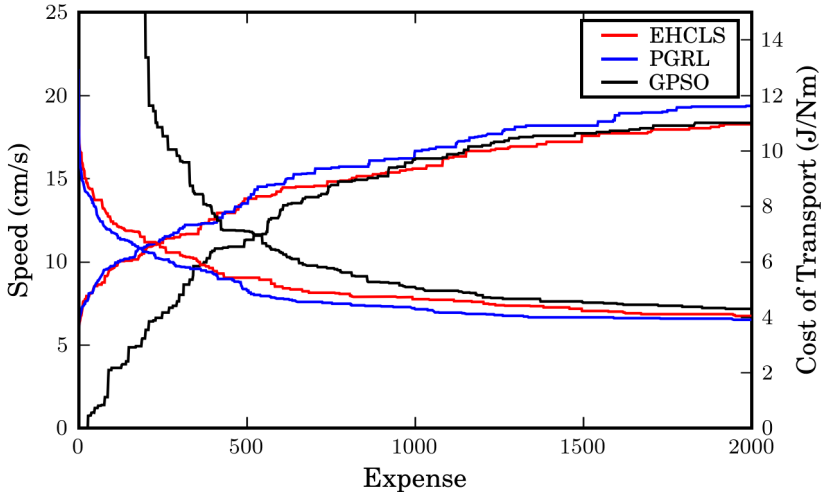


Figure 6.11: The evolution of the walk speed and efficiency for the three optimisation algorithms evaluated. The median speed and efficiency are plotted for each algorithm.

Table 6.3: Comparison of Algorithms

Algorithm	Speed (cm/s)	Cost of Transport (J/Nm)	Falls (%)
EHCLS	18.3 (0.8)	4.04 (0.31)	16.3
PGRL	19.4 (0.6)	3.89 (0.18)	8.7
GPSO	18.3 (0.8)	4.29 (0.28)	11.9

Table 6.3 shows the median speed and efficiency of the final optimised walks from each combination of fitness function and parameter space for the three algorithms. It also includes the variance between the simulation runs in parentheses. The final column is the median average fall percentage and provides a measure of the stability of a gait. For example, the EHCLS has a fall percentage of 16.3%. This means that on average, the best gaits selected by the EHCLS, have a 16.3% chance of causing the robot to fall while traversing the optimisation path. The fall percentage was measured for each of the final optimised gaits by using each set to complete the path repeatedly over a 20 minute period.

The results show the PGRL algorithm performs better in terms of improving the walk speed and efficiency by a significant margin. PGRL also has less variation between the fitness of the optimised gaits and has a lower fall percentage, implying that the gaits are more stable than those selected by either EHCLS or GPSO. This improvement in stability is achieved even though each algorithm is using the same implicit stability measure in the fitness functions.

The observed improvement in stability of the walks selected by PGRL stems from the averaging of several nearby policies to estimate the gradient. As PGRL uses the outcome of several walk evaluations to calculate the next set of parameters it avoids areas where the walks may be fast, but are either unreliable, or sensitive to small changes in the walk parameters that result in instability.

In contrast, EHCLS has a tendency to select unstable walk parameters as an unstable walk may be able to complete the optimisation path occasionally. The EHCLS algorithm effectively tests a single set of parameters repeatedly until an improvement is achieved. This tends to allow the algorithm to drift toward walks which are fast but erratic, because the walk need only complete the path a single time to be used to calculate the next set of parameters.

The GPSO has the disadvantage of starting from an initial distribution of walk parameters where the majority of the parameters are unstable. Hence, the algorithm is heavily penalised until all of the particles converge toward stable walks. This is evident in Figure 6.11 where it is not until an expense of 1000 that the GPSO algorithm begins to produce comparable walks.

The EHCLS, PGRL and GPSO algorithms were able to complete 540, 1000 and 660 iterations, on average, before using all of the available expenditure. Therefore, it is possible that the ratio between the cost of a successful iteration and a fall may influence the comparison. However, the effect was observed to be reasonably small, when the comparison was performed using ratios of 1:7 and 1:13; similar results were achieved, as shown in Figure 6.12.

Finally, Figure 6.11 shows that the choice of terminating expenditure has little impact on the comparison. Provided the allocated expenditure exceeds 1000, all three algorithms continue to improve the walk at approximately the same rate, with PGRL performing slightly better.

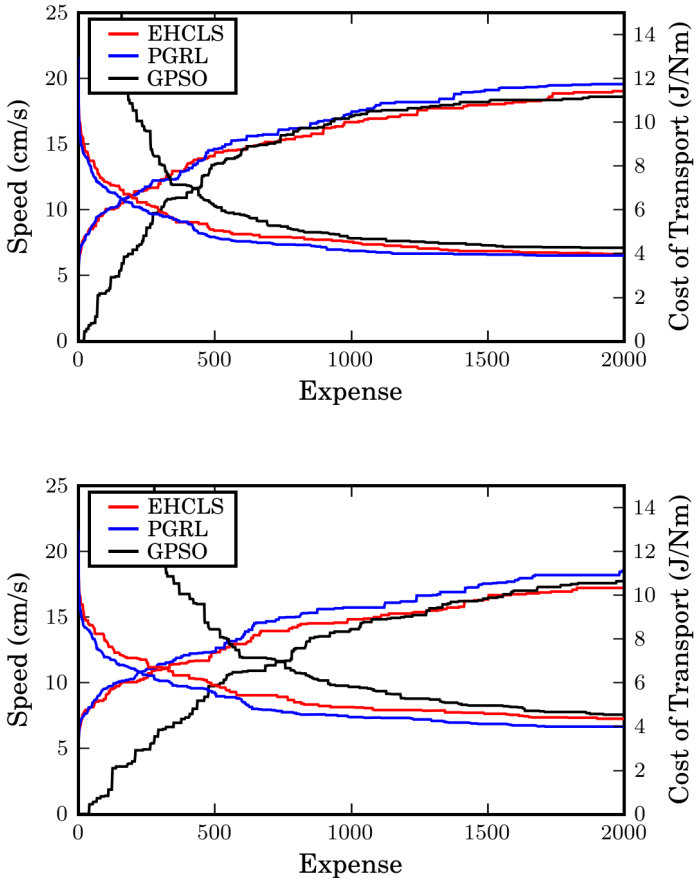


Figure 6.12: The effect of varying the expense ratio between successful and unsuccessful trial. The top figure uses a ratio of 1:7 and the bottom figure uses a ratio of 1:13.

6.9 Comparison of Fitness Functions

Recall that Figures 6.5 to 6.10 also show the specific performance of the fitness functions with each combination of algorithm and parameter space. The figures show that each fitness function performs well in improving walk speed and efficiency. This demonstrates that the careful design of the functions $\mathcal{F}_S$, $\mathcal{F}_C$ and $\mathcal{F}_F$ to have similar numerical properties has enabled the algorithms to use each function effectively without repeating the meta–optimisation.

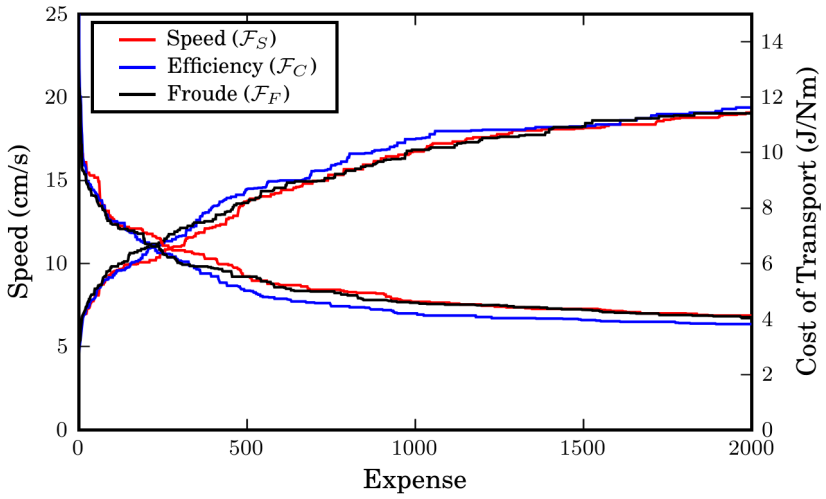


Figure 6.13: The evolution of the median walk speed and efficiency achieved with each of the three fitness functions.

Figure 6.13 shows the evolution of the median speed and efficiency of the walk performance for each fitness function as a function of expense. The figure shows that the evolution of the speed–based ($\mathcal{F}_S$) and Froude–based ($\mathcal{F}_F$) fitness functions are almost identical. The efficiency–based fitness ($\mathcal{F}_C$) performs better at improving the speed in the early stages of the optimisation and performs significantly better in reducing the cost of transport.

Table 6.4 shows the median speed, efficiency and fall percentage of the final optimised walks for each fitness function. The results in the table demonstrate that the efficiency–based fitness function ($\mathcal{F}_C$) outperforms the other two in every respect. Interestingly, it even outperforms the speed based fitness function at improving the walk speed. The NAO has a high idle energy consumption, so the efficiency can be significantly improved by simply completing the path faster. This property makes optimising the efficiency particularly effective at simultaneously improving the speed. Furthermore, humans also have a high basal energy consumption [87], and there is evidence to support the proposition that human evolution has used a cost of transport based fitness function [120].

The cost of transport based fitness function also selects gaits that are signif-

Table 6.4: Comparison of Fitness Functions

Function	Speed (cm/s)	Cost of Transport (J/Nm)	Falls (%)
Speed	18.3 (0.8)	4.23 (0.34)	15.4
Efficiency	18.9 (0.6)	3.88 (0.19)	8.0
Froude	18.7 (0.8)	4.19 (0.24)	13.7

Table 6.5: Comparison of Parameter Spaces

Space	Speed (cm/s)	Cost of Transport (J/Nm)	Falls (%)
$\mathcal{W}_p \cup \mathcal{W}_s$	18.1 (0.7)	4.07 (0.29)	9.7
$\mathcal{W}_p$	19.3 (0.8)	4.14 (0.22)	15.1

icantly more stable. The gaits are forced to exploit the natural dynamics of the robot so as to become more efficient. Additionally, the walks become smoother as unnecessary jerk is reduced to conserve energy. Both these features make the walks inherently more stable.

6.10 Comparison of Parameter Spaces

Figure 6.14 and Table 6.5 compare the median speed and efficiency from all of the combinations. Table 6.5 shows that the gaits produced with additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$) are slower. However, they are more efficient and more stable. Lowering the stiffness in each joint through the optimisation process has the effect of reducing both the maximum torque exerted by the motor and the controller gain. This allows the robot to 'settle' into a more efficient gait, rather than attempting to rigidly follow joint trajectories produced by an imperfect walk engine.

We found in Section 6.8 that PGRL was the most suitable algorithm, and in Section 6.9 that $\mathcal{F}_C$ was the best fitness function. If we consider only this combination we find that the median speed of the final select walks drop from 19.3cm/s to 18.1cm/s when the joint stiffness parameters are added. However, adding the stiffness parameters improves the median fall percentage from 2.1% to a perfect zero. That is, the 12 final optimised walks selected by each simu-

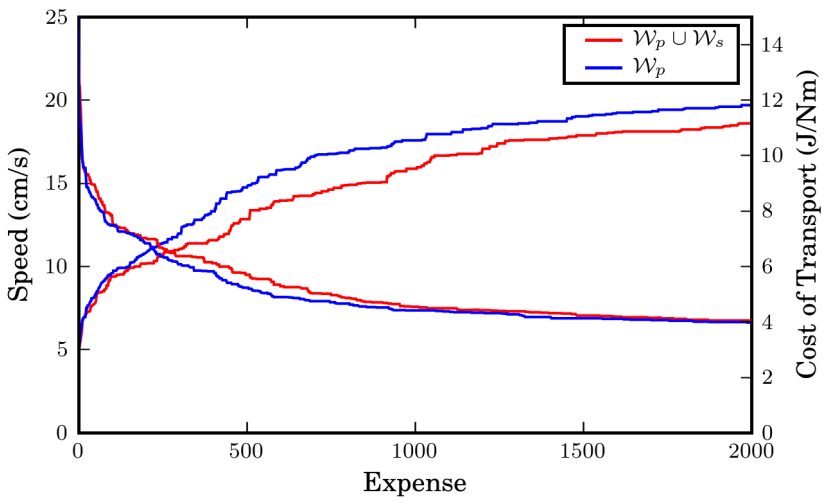


Figure 6.14: The evolution of the median walk speed and efficiency achieved using the parameter spaces $\mathcal{W}_p \cup \mathcal{W}_s$ and $\mathcal{W}_p$.

lation instance with PGRL, $\mathcal{F}_C$ and the additional stiffness parameters, had a zero fall percentage when tested over a 20 minute period.

This is an important result and means that if this combination is used to perform the optimisation process only a single time, it is highly probable that the final optimised walk will never fall. This is ideal for real hardware where it is only practical to run the optimisation process a single time.

The reader may recall that a significant improvement in speed was achieved in Chapter 5 when the additional stiffness parameters were added to the optimisation on a physical NAO. This is in contrast to the small reduction in speed observed here in simulation. This discrepancy is primarily due to limitations of the simulation itself, and not due to the differences in walk engines.

The physical NAO uses very high controller gains. Similarly high gains in the simulator result in simulation–instability, consequently the initial set of joint stiffness parameters used for optimisation were much lower than the default values used in hardware. Furthermore, the accuracy of the simulation of the joint stiffness is limited, the low–level controllers are simply proportional and the simulated motors have 100% efficiency at all speeds. Thus, the effect of a reduction in stiffness is not as significant in simulation.

The simulated robot is only an approximation of the actual robot, where the physical modelling in the simulator is not ideal. In particular, there are no self–collisions and each joint is connected together via a lightly damped. The effect is that it is much easier to perform the walk optimisation in simulation as the simulated robot appears to be much more stable. Thus, the use of reduced stiffness is not required to improve the speed of the walk in simulation, the opposite of that observed on the physical robot.

Combining the improvement in stability and efficiency observed in simulation in this section with the improvements in speed, efficiency and stability observed previously, in hardware, in Chapter 5 a strong case for the use of additional joint stiffnesses in the walk optimisation can be made. Consequently, the additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$) are included in the final meta–optimised walk optimiser in the next section.

6.11 Application to the Physical NAO

In summary, Sections 6.6–6.10 used a simulator to design a general walk optimiser for application to physical robots. In Section 6.6 we selected the best parameters for the optimisation algorithm. In Section 6.8 we selected PGRL as the best algorithm on the basis of the superior speed, efficiency and stability of the optimised walks. In Section 6.9 an efficiency based fitness function performed best in every aspect considered. Finally, in Section 6.10 we saw that additional stiffness parameters improved stability and efficiency, and in Chapter 5 we saw a significant improvement in speed in hardware.

Based on these results, the combination of PGRL with $\mathcal{F}_C$ and the additional stiffness parameters was applied to a physical NAO to verify the effectiveness of the meta–optimised walk optimisation system. Additionally, the optimisation path designed in the simulator, shown in Figure 6.2, and the stress measure (6.1) were used for the optimisation on the physical robot.

Recall that the physical NAO uses the walk engine provided by Aldebaran and that the default walk parameters were used as a starting point for the optimisation.

The results of the optimisation process are shown in Figure 6.15. The optimiser significantly improves the performance of the walk compared to the default settings. The average speed was improved from 7.0cm/s to 11cm/s,

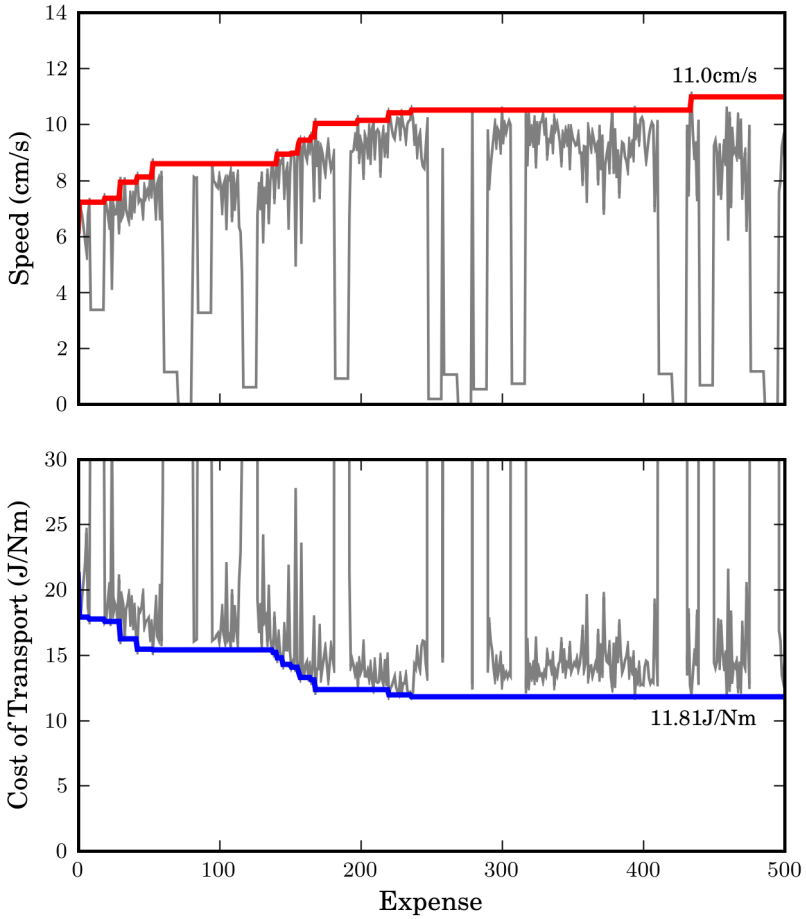


Figure 6.15: The results for the meta–optimised PGRL with efficiency based fitness function $\mathcal{F}_C$ and additional stiffness parameters ($\mathcal{W}_p \cup \mathcal{W}_s$) on a physical NAO. The thick lines represent the best speed and efficiency. The thin lines indicate the individual speed and efficiency for each set of walk parameters trialled. The short sections where the speed of an individual trial is low are the result of the robot falling and a penalty being applied.

an improvement of 57%. The efficiency is also improved, from 16.9J/Nm to 11.8J/Nm, an improvement of 30%. Furthermore, the selected walk is very stable, having a zero fall percentage over 30 iterations of the optimisation path.

There is a large difference between the optimised walk speed on the physical

robot and in the simulator. In addition to the limitations of the simulator discussed previously, we note that the load on the simulated joints does not effect their maximum speed. Thus, the simulated robot is capable of moving its limbs much faster than the physical robot, resulting in a higher walking speed.

The improvement on the physical robot is also much smaller than that observed in the simulator. The default walk parameters for the Aldebaran walk have presumably been optimised in some sense before the release of the walk engine. In contrast, the starting point used for the simulated walk was selected quickly and not optimised in any manner.

It is informative to compare the results of this chapter to those in Chapter 5. In Chapter 5 we optimised only the forward walk along a straight path, achieving a speed of 13.9cm/s. In this chapter, we optimise an omnidirectional walk over a complex path and achieved an *average* speed of 11.0cm/s. The peak forward speed of the optimised walk from this chapter was 20cm/s, when this speed is compared to the speed from the previous chapter, the effectiveness of the proposed system becomes clear. Similarly, the efficiency while forward walking achieved in this chapter was 4.9J/Nm, compared to 5.8J/N in the previous chapter.

The significant improvement in performance compared to Chapter 5 also demonstrates the utility of the simulation results. An optimiser was designed in Sections 6.6–6.10 in simulation then, without modification, the optimiser was applied to the physical robot. It appears that the general conclusions drawn from the results in the simulator apply to the physical robot, including the selection of algorithm parameters and the selection of algorithm, fitness function and parameter space. Furthermore, the use of the optimisation path also resulted in an optimised walk with excellent omnidirectional capabilities and the stress measure appears to have reduced the number of falls during the optimisation procedure.

The two best walk engines for the NAO circa 2010 were the NAO Devils' engine [55] and B–Human's engine [56]. Both of these walk engines are much more advanced than the Aldebaran walk engine optimised throughout this thesis, the walk engines produce dynamically stable gaits and use sensor feedback to respond to external perturbations.

The NAO Devils' walk reached speeds close to 45cm/s [167], however, this speed was not achievable during a game due to problems with overheating. One of the advantages of optimising the efficiency of the walk is that it reduces the energy consumption of the motors, thereby reducing their operating temperatures. The optimised walk presented in this chapter is capable of continuous walking without overheating.

The B–Human walk had a forward walk speed of approximately 25cm/s in competition [168]. The forward walk speed achieved in this Chapter was 20cm/s, despite the more limited walk engine. Furthermore, here we optimise the walk over a complex path and achieve an average speed of 11.0cm/s. It is unclear what the average speed of the b–human walk would be over a similar path, however, their robot's rarely reach there maximum speed during a match as they constantly slow to change directions. Thus, the optimised walk of this chapter should compare favourably.

6.12 Conclusion

The Policy Gradient Reinforcement Learning algorithm performed better in terms of improving the walk speed and efficiency and does so more consistently. Furthermore, the PGRL algorithm significantly outperforms the EHCLS and GPSO in selecting gaits which are stable.

The cost of transport based fitness function outperformed both the speed and Froude–number based functions in every respect, especially in terms of the selected gait stability. The addition of joint stiffnesses to the parameter space also improved the efficiency and stability of the selected gaits. The results demonstrate that a significant improvement in walk stability can be achieved through careful selection of an algorithm, fitness function, and parameter space.

From the results we conclude that the best algorithm–fitness–space combination for a humanoid robot walk optimisation system is PGRL with an efficiency based fitness function and a parameter space with additional stiffness parameters. In the simulator, all of the walks selected by this combination had a zero fall percentage, while being extremely fast and efficient.

To support the results from the simulator we applied the proposed walk optimisation system to a physical NAO, without modification. The optimiser

successfully improved the walk speed by 57% and walk efficiency by 30%. The walk selected using this procedure was used at the 2010 RoboCup competition. We found the walk to have a comparable performance to many other teams who were using more advanced walk engines. This was especially true when the robots were required to walk in an omni–directional manner. Furthermore, throughout the competition the walk was observed to be exceptionally stable, only falling after repeated contact with other robots.

Chapter 7

Walk Optimisation with Redundant Fitness Functions

Contents

7.1 Introduction 118

7.2 Equipment and Method 119

7.3 Opposition–based PGRL with Redundant Fitness Functions 121

7.3.1 Opposition–based Policy Generation 121

7.3.2 Use of Redundant Fitness 121

7.4 Applications of the Improved PGRL Algorithm . . 124

7.5 Conclusion 129

7.1 Introduction

This chapter discusses an extension of the PGRL algorithm to include the use of redundant fitness functions to improve convergence speed and assist in the escape from local extrema. The PGRL algorithm is also modified such that the policies used to estimate the gradient are generated in opponent pairs. The improved algorithm is validated using a simulated humanoid robot and two physical robots, the NAO and the DARWIN-OP. Two different physical humanoid robots are used to demonstrate the generality of the proposed improvements.

In Section 6.8 we found PGRL to be the best performing optimisation algorithm of those considered. However, the PGRL algorithm is only a local optimiser and is consequently prone to getting trapped in local extrema. One strategy to deal with local extrema is to reset the algorithm using a new starting point. This approach is difficult to apply to humanoid walk optimisation, because another set of stable parameters is generally difficult to find. Frequent resets result in the algorithm reducing to a random search.

An alternative approach is to use the concept of safe redundancy [169, 74], where instead of restarting the algorithm, the fitness function is modified to escape the local extremum. In particular, the fitness function is replaced by a different fitness function, that shares the same global extremum, each time the optimisation process reaches a local extremum. The concept of safe redundancy can be used in the more general case where a set of fitness functions have similar global extrema. The differences in local extrema between each of the different fitness functions still help local optimisers to move toward a more global extremum.

In Section 6.4 we proposed three fitness functions specifically designed to have similar numerical properties. This design allows the fitness functions to be interchanged without adjusting the algorithm. The three fitness functions were compared in Section 6.9, where each performed well at improving the walk speed and efficiency. This suggests that the fitness functions have similar global extrema and are suitable for use in the application of the safe redundancy concept to the PGRL algorithm.

The second modification to the PGRL algorithm is to incorporate opposition–based machine learning, which involves evaluating a particular policy and also

evaluating its 'opposite' [170, 171]. This idea can be applied to extend many existing optimisation algorithms to improve convergence speed.

The remainder of this chapter, firstly describes the additional robot platform used for validation of the proposed improvements. Next, the improved algorithm is presented, followed by the results of its application to three humanoid robot platforms.

7.2 Equipment and Method

The equipment and method used here is identical to that used in the previous chapter with the addition of an extra physical robot platform, the DARWIN-OP [26]. The reader is referred to Section 6.2 for more details on the simulated and physical NAO (v3.2), and the optimisation process.

Figure 7.1 shows the DARWIN-OP alongside the NAO robot platform. The DARWIN-OP was selected as the second physical platform due to its availability from the 2012 NUbot humanoid soccer team. The two robots have similar sizes, both being kid–sized humanoid robots, and have comparable degrees of freedom. However, for their heights, the robots have different sized feet, the DARWIN-OP having feet 25% smaller, and different Body Mass Indexes, the DARWIN-OP being 15% lower.

The walk engine used on the DARWIN-OP was the open–source walk engine of Robotis [172] supplied with the robot. The initial walk parameters were manually selected as the default walk parameters were unstable, hence the robot was unable to complete the optimisation path. Table 7.1 shows the walk parameters for the DARWIN-OP walk engine. For comparison, the table also shows the parameters for the walk engines of the physical and simulated NAO.

In addition to the optimisation process discussed in Section 6.2, the same meta–optimised parameters for the PGRL algorithm found in Section 6.6 are used in this chapter. Furthermore, the fitness function selected in Section 6.9 and the parameter space selected in Section 6.10 are also used without modification. Thus, the results of this chapter serve as a validation of the improved PGRL algorithm and also demonstrate the generality of the walk optimiser designed in the previous chapter.

Figure 7.1: The simulated NAO robot, the physical NAO robot and the DARWIN-OP robot. The two physical robots are pictured with overhead vision markers attached.

Table 7.1: Walk Parameters

	Sim. NAO		Phys. NAO		DARWIN-OP	
Parameter	Min	Max	Min	Max	Min	Max
Velocities (cm/s)	[0,0,0]	[70,70,2]	[0,0,0]	[30,30,2]	[0,0,0]	[30,20,2]
Accelerations (cm/s/s)	[0,0,0]	[140,140,4]	[0,0,0]	[140,140,4]	[0,0,0]	[140,140,6]
StepFrequency (Hz)	1	5	1.6	4.5	1.6	4.0
StepHeight (cm)	0	8	0.5	6	1.0	4
ZMPStatic	0	1	-	-	-	-
ZMPOffset (cm)	0	7	-	-	-	-
SwayLeftRight (cm)	-	-	-	-	1.5	3.5
SwayUpDown (cm)	-	-	-	-	0	0.8
SensorGain	[0,0]	[0.2,0.2]	-	-	[0,0,0,0]	[1,1,1,1]
SensorSpring	[0,0]	[1000,1000]	-	-	-	-
DSFraction	0.1	0.8	-	-	0.1	0.5
HipHack (rad)	0	0.3	0	0.2	-	-
FootLift (rad)	-0.4	0.4	-	-	-	-
TorsoPitch (rad)	-0.2	0.4	-0.15	0.15	-0.15	0.15
TorsoHeight (cm)	25	32	27	32	18	24
HipStiffness (%)	[30,30]	[100,100]	[30,30]	[100,100]	[30,30]	[100,100]
YawStiffness (%)	30	100	30	100	30	100
KneeStiffness (%)	30	100	30	100	30	100
AnkleStiffness (%)	[30,30]	[100,100]	[30,30]	[100,100]	[30,30]	[100,100]

7.3 Opposition–based PGRL with Redundant Fitness Functions

7.3.1 Opposition–based Policy Generation

To extend PGRL to include opposition–based learning we need only define what constitutes an opponent policy. We define an opponent policy as being a policy with opposite perturbations in each dimension. For example, given the policy

$$\{\theta_0 + \epsilon, \theta_1 + 0, \theta_2 + \epsilon, \theta_3 - \epsilon\},$$

the opponent policy would be

$$\{\theta_0 - \epsilon, \theta_1 + 0, \theta_2 - \epsilon, \theta_3 + \epsilon\},$$

where the sign of the perturbation has been changed in each dimension. This modification is implemented by first generating $N/2$ policies in the usual manner and then generating the remaining policies to be their opponents.

One of the motivations for the inclusion of opponents in the policy generation was to balance the number of perturbations in each direction. In the original formulation the expected number of policies with $\theta_i - \epsilon$, θ_i or $\theta_i + \epsilon$ are equal. However, with a small number of policies per iteration the number in each direction is frequently unbalanced and occasionally a particular direction will have no policies. The opposition–based policy generation guarantees that the number of policies with $\theta_i - \epsilon$ and $\theta_i + \epsilon$ are equal in each iteration.

7.3.2 Use of Redundant Fitness

To incorporate the concept of safe redundancy into the PGRL algorithm, the three fitness functions outlined in Section 6.4 are used. The number of trials between improvements is used as an indicator for when a local maximum is reached. When there are no improvements for a fixed iteration count, the fitness function is switched.

Figure 7.2 sketches an example of how the three different fitness functions are used together in a single dimensional case. The figure shows the fitness of the parameter as measured by each of the three fitness functions. In this

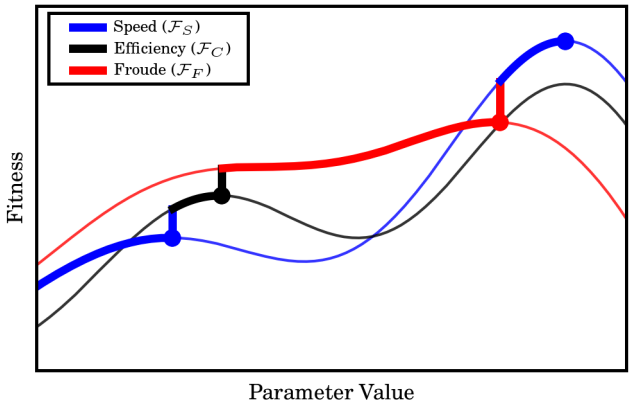


Figure 7.2: An example of a one–dimensional hill climb algorithm operating on three fitness functions. Local maximums reached for each each fitness function are shown with a circular marker.

example, the three fitness functions have similar, but not identical, global maxima. The results of Section 6.9 suggest that this situation is the case with the three fitness functions proposed in Section 6.4.

Assume, the algorithm begins from the left and moves right, following the gradient of the fitness in the first function ($\mathcal{F}_S$). As the parameter is changed from left to right a local maximum is reached, indicated by the circular marker. The optimiser detects that it has stalled and switches to the next available fitness function ($\mathcal{F}_C$) to escape from the local extremum. Similarly, the second fitness function is used until another local maximum is reached and then the fitness function is switched again. This process is repeated until a cycle is reached where the optimiser is trapped in local extrema in all of the available fitness functions.

Algorithm 4 shows the improved PGRL algorithm, incorporating opposition–based learning and redundant fitness functions. The improved PGRL algorithm has one additional parameter, L, specifying the iteration count at which a stall is detected and the fitness function is switched. The same meta–optimisation procedure described in Section 6.6 was used to select $L = 25$, with the remaining PGRL parameters left unchanged.

Algorithm 4 Opponent–based PGRL with Redundant Fitness Functions

$\theta = InitialParameters$
loop
  $\{\theta\}_1^N = \texttt{generatePolicies}(\theta, \epsilon)$
  $\mathcal{F} = \texttt{calculateFitness}(\{\theta\}_1^N)$
  for $i = 0$ to D **do**
    $Avg^i_{+\epsilon} \leftarrow$ average fitness for all θ^k that have a positive perturbation in dimension i
    $Avg^i_0 \leftarrow$ average fitness for all θ^k that have no perturbation in dimension i
    $Avg^i_{-\epsilon} \leftarrow$ average fitness for all θ^k that have a negative perturbation in dimension i
    if $Avg^i_0 > Avg^i_{+\epsilon}$ and $Avg^i_0 > Avg^i_{-\epsilon}$ **then**
      $A_i = 0$
    else
      $A_i = Avg^i_{+\epsilon} - Avg^i_{-\epsilon}$
    end if
  end for
  $A = \eta \tanh(A) \cdot \texttt{range}(\theta)$
  $\theta = \theta + A$
  if `improvement()` **then**
    $StallCount = 0$
  else
    $StallCount+ = 1$
  end if
  if $StallCount > L$ **then**
    `calculateFitness` $\leftarrow$ next redundant fitness function
  end if
end loop

---

where

$\{\theta\}_1^N$ = set of N walk parameter sets
D = dimension of θ
`generatePolicies(a,b)` = generates N walk parameter sets such that $\theta^k = \{\theta_1 + \delta_1^k, \ldots, \theta_D + \delta_D^k\}$. The first $N/2$ δ_i^k are chosen randomly from $\{-\epsilon_i, 0, \epsilon_i\}$ where $\epsilon_i = \epsilon \cdot \texttt{range}(\theta_i)$. The remaining $N/2$ are chosen such that $\delta_i^k = \delta_i^{k-N/2}$.
A = adjustment vector $\texttt{range}(\theta) = \{\theta_{max} - \theta_{min}\}_1^D$ the size of the parameter space for each dimension D

with the following parameters previously meta–optimised

η = specified the step size
ϵ = size of the perturbation applied to each dimension
N = number of policies

and the following additional parameter to be meta–optimised

L = limit at which a stall is detected

7.4 Applications of the Improved PGRL Algorithm

The performance of the standard and improved PGRL algorithms were compared on both the simulated and physical NAO. Figures 7.3 and 7.4 show the results for the simulated and physical robots, respectively. The improved algorithm was also applied to the DARWIN-OP to validate the effectiveness of the algorithm on a different humanoid robot platform. The results of the application to the DARWIN-OP are shown in Figure 7.5.

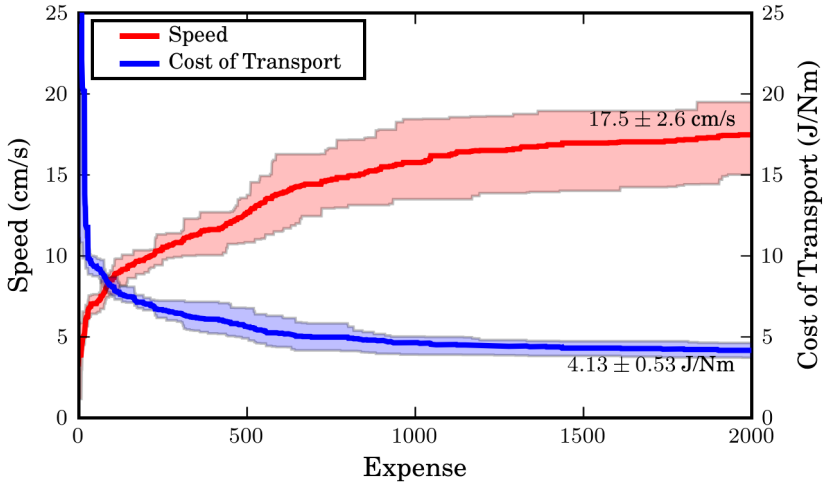
In simulation we are able to run the experiment repeatedly to measure the variation between trials of each algorithm. Figure 7.3 shows the median speed and efficiency over 12 simulations, as well as the 25th and 75th percentiles. It is clear from the figure that the PGRL with redundant fitness functions performs significantly better. The speed and efficiency are improved 15% and 9%, respectively, over the standard PGRL algorithm and have much less variation between trials.

On the physical NAO, the difference observed between the two PGRL variants was smaller. The PGRL with redundant fitness functions improved the speed and efficiency 4% and 8%, respectively, as compared to the standard PGRL. The overall improvement in speed and efficiency achieved by the improved PGRL algorithm compared to the default walk parameters was 61% and 38%, respectively.

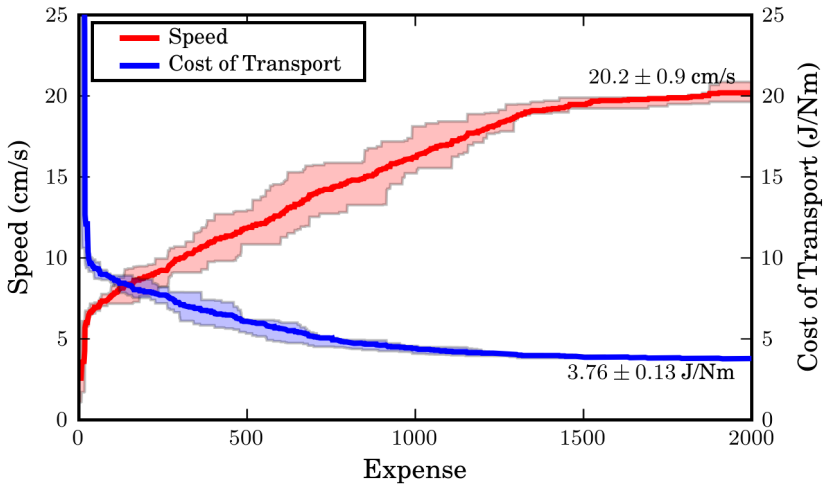
Figure 7.4(b) shows the locations where the optimisation algorithm detects progress has stalled and switches to another fitness function. There is a subsequent improvement in the speed and efficiency of the walk after each fitness function switch. Furthermore, it appears the stall threshold of 25 iterations is set low enough that the redundant fitness functions also improve the convergence speed of the PGRL algorithm. When the algorithm is temporarily stalled, in an area that has an approximately flat gradient, switching to a different fitness function quickly improves the optimisation.

One of the major differences between the simulated and physical systems is that the amount of noise in the physical system is greater. Noise can have the tendency to assist optimisers to escape from local extrema and may explain the smaller improvement observed in hardware as compared to that in simulation.

The PGRL with redundant fitness functions was also applied to the walk optimisation on the DARWIN-OP platform. Figure 7.5 shows that the optimiser

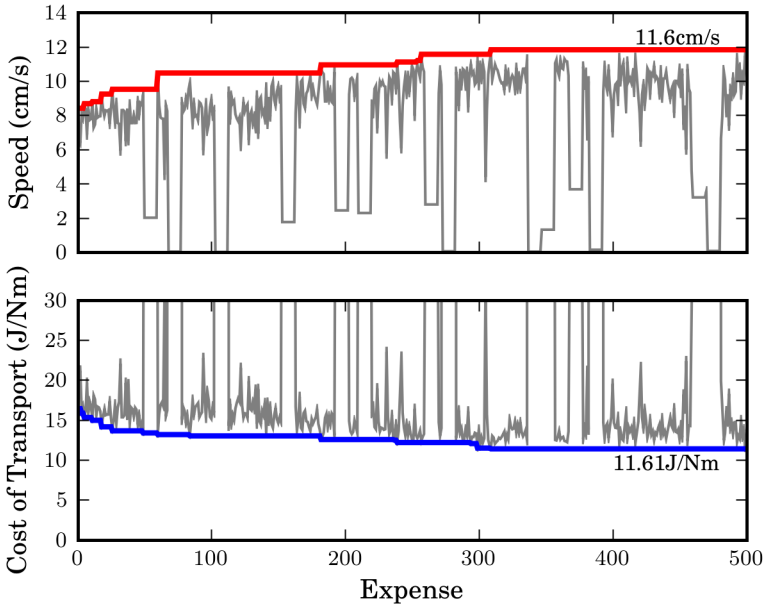


(a) The standard PGRL algorithm

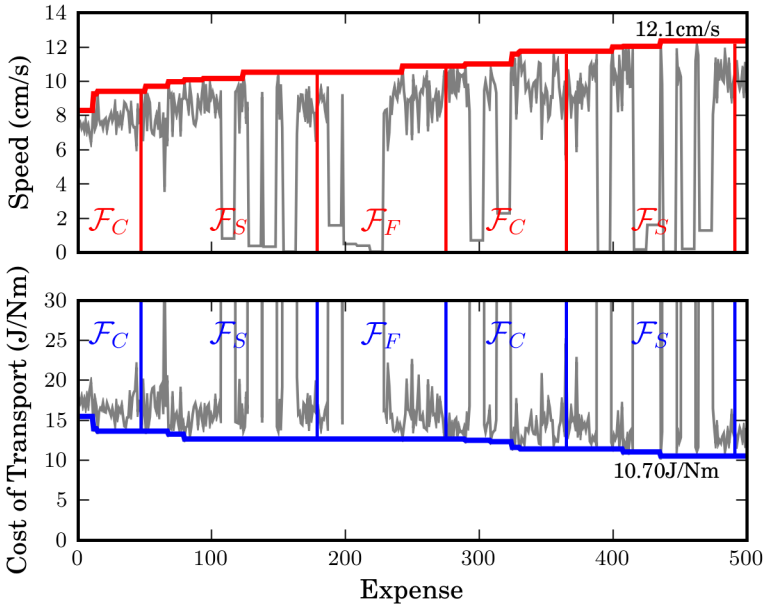


(b) The PGRL algorithm with redundant fitness functions

Figure 7.3: The median speed and efficiency as a function of expense for the two PGRL variants in simulation. The shaded regions are bounded by the 25th and 75th percentiles for the speed and efficiency.



(a) The standard PGRL algorithm



(b) The PGRL algorithm with redundant fitness functions

Figure 7.4: The results of the walk optimisation on the physical NAO using the two PGRL algorithms. The thick lines are the best speed and efficiency as a function of expense. The thin lines are the values for each individual trial. The current fitness function in use is indicated in (b).

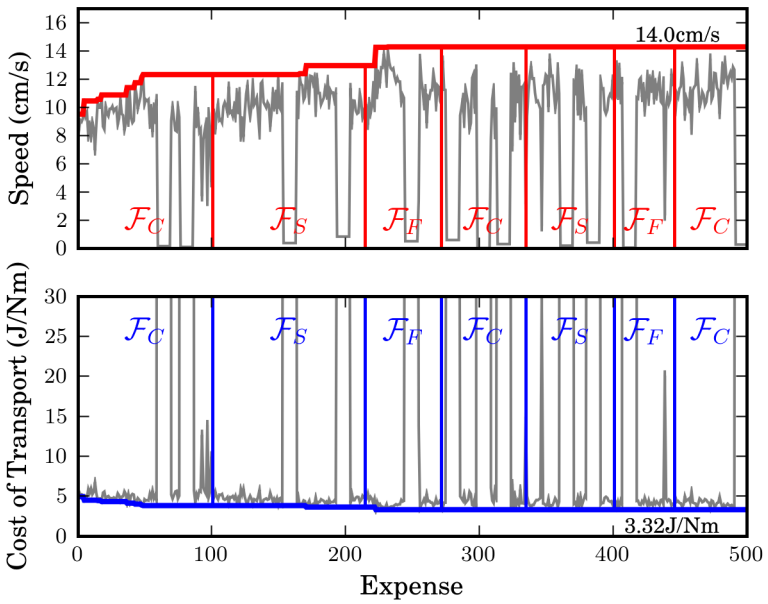


Figure 7.5: The results of the walk optimisation on the DARWIN-OP using the PGRL with redundant fitness functions. The thick lines are the best speed and efficiency as a function of expense. The thin lines are the values for each individual trial. The fitness function labels indicate the current function being used.

performs well, improving the speed and efficiency of the walk by 65% and 40% from the starting point, respectively. The improvement in performance achieved on the DARWIN-OP is slightly higher than that achieved on the physical NAO.

The forward walking speed of the NAO achieved in this chapter was 20.5cm/s. This value is approximately the same as that achieved in Chapter 6. It is insightful to note that this speed appears to approach the maximum speed possible using the Aldebaran walk engine on the NAO. Further attempts at small increases in maximum speed result in the walk engine entering a fault state, where motion patterns are no longer produced, rendering the NAO unable to walk. This limitation highlights the impact of the walk engine on the performance of the optimised walk.

Through the application of the optimiser presented in this chapter the DARWIN-OP achieved a maximum forward speed of 23cm/s. As the DARWIN-

OP is quite a new humanoid robot there are few examples of optimised walks in the literature. Robotis quote the DARWIN-OP having a forward walking speed of 24cm/s [149]. However, as mentioned previously, the Robotis set of walk parameters failed to complete the optimisation path and appeared to be unstable. To date the fastest walk for the DARWIN-OP appears to be 36cm/s [173]. This higher speed was achieved with a much more advanced walk engine than that which was optimised in this chapter.

The significant difference in performance achieved on the two humanoid robots in this chapter highlights the influence of the robot hardware. The DARWIN-OP is equipped with a very simple walk engine when compared to that of the NAO, lacking a formal method for walk pattern generation. However, the DARWIN-OP is significantly lighter, especially in the legs, allowing a higher speed and efficiency to be achieved.

The successful application of the optimiser to another physical robot demonstrates the platform independence of the proposed improvements to the PGRL algorithm. Furthermore, the optimiser from Chapter 6 has been used unmodified, apart from the improved PGRL algorithm, in this chapter. Thus, the favourable results on the second physical robot also demonstrate the generality of the design in Chapter 6 and further suggest the principles developed in the simulator are generally useful.

Additionally, the same meta–optimised algorithm parameters selected in Section 6.6 and the three fitness function proposed in Section 6.4 are used on all three platforms, demonstrating their generality. The additional joint stiffness parameters are also used successfully on the DARWIN-OP to improve the efficiency and stability of the walk. The same optimisation path shown in Figure 6.2 is also used for all three robot platforms and produces stable omnidirectional walks in each case.

The procedure for using redundant fitness functions outlined in this chapter is general enough to be applied to any local optimisation algorithm. The only requirement is the availability of a set of fitness functions that have similar global extrema, but different local extrema. For example, the fitness function switching could be applied to the EHCLS algorithm by replacing the reset phase in that algorithm with the fitness function switching proposed here.

7.5 Conclusion

An improved PGRL algorithm was proposed that includes opposition–based learning and redundant fitness functions. The standard and improved PGRL algorithms were applied to the optimisation of existing walk engines in both simulation and hardware. It was found that the improved PGRL algorithm performed better in both cases.

In simulation a further improvement of 15% in speed and 10% in efficiency was achieved using the improved PGRL over that of the standard algorithm. On the physical NAO the improvements were smaller, a further 4% in speed and 8% in efficiency as compared to the standard PGRL algorithm. This gives an overall improvement, as compared to the default settings, of 61% and 38% in speed and efficiency, respectively. The additional noise in the physical system is likely to be the cause of the smaller improvement in hardware.

The improved algorithm was also applied to the DARWIN-OP robot. An improvement in speed and efficiency of 65% and 40%, respectively, over the initial walk parameters was achieved. The successful application of the optimiser to the new platform demonstrates the effectiveness of the improved algorithm, as well as the generality of the optimiser designed through meta–optimisation in Chapter 6.

Chapter 8

Gait–Phase Dependent Joint Stiffnesses

Contents

8.1 Introduction 131

8.1.1 Phases of the Gait Cycle 132

8.2 Phase Dependent Stiffness with Fixed Traditional Walk Parameters 134

8.2.1 Equipment and Method 134

8.2.2 Results 140

8.3 Optimisation of Phase Dependent Stiffness and Traditional Walk Parameters 140

8.3.1 Equipment and Method 140

8.3.2 Results 142

8.4 Discussion 144

8.4.1 Phase Dependent Stiffness with Fixed Traditional Walk Parameters 144

8.4.2 Phase Dependent Stiffness with Variable Traditional Walk Parameters 145

8.4.3 Robot Tracking 146

8.5 Conclusion 147

8.1 Introduction

In this chapter we extend the results of Chapters 5 and 6 with respect to the optimisation of joint stiffnesses. In the earlier chapters we specified the stiffness for each joint individually. Here, in addition to specifying the stiffnesses for joints individually, we specify stiffnesses for each phase of the gait cycle.

The specification of stiffness as a function of gait phase is a form of gain scheduling. In the literature gain scheduling has been applied to the control of a humanoid robots. In simulation, Kim et al. [174] used the location of the Zero Moment Point to modify PID gains for hydraulically actuated joints to improve trajectory tracking. In another example, two sets of PD gains for each joint have been used to reduce vibration and improve tracking accuracy of a forward walk [175, 176]. In this example the two sets of feedback parameters were interchanged depending on whether the leg was supporting the weight of the robot or not.

The research presented in this chapter extends the literature described above in several ways. Firstly, the gait cycle is split into four phases: stance, push, swing and impact. The gait phase is measured in real–time and used to specify the stiffness for each of the four phases. Secondly, the stiffness parameters are selected through a rigorous optimisation process to improve the more practical performance measures of walk speed and efficiency. Furthermore, we apply the gait–phase dependent stiffness to an omnidirectional walk engine on a physical robot.

The remainder of this chapter is organised as follows; first, the phases of the humanoid robot gait cycle and the system used to measure the current phase are discussed. Next, two applications of gait–phase dependent stiffness are presented. Section 8.2 presents the first application where phase dependent joint stiffnesses were used with a walk engine whose traditional walk parameters were fixed, hence illustrating the benefits of modifying only the stiffness parameters. This first application was prior to the introduction of the overhead ssl–vision tracking system, consequently a custom laser–scanner based system was employed which is also briefly described.

The second application of gait–phase dependent stiffness is presented in Section 8.3 and involved the simultaneous optimisation of both the traditional

walk parameters and the phase dependent stiffnesses. This later application made use of the improved optimisation system developed in Chapters 6 and 7.

Finally, the results of both applications of gait–phase dependent stiffness are discussed and a comparison of the two robot tracking systems is presented.

8.1.1 Phases of the Gait Cycle

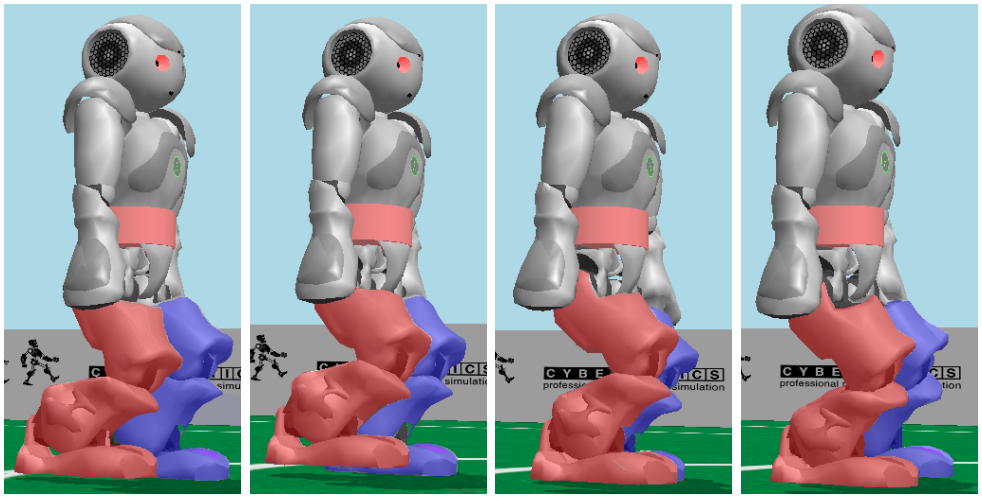
The gait cycle of each leg is split into four phases; stance, push, swing and impact. Essentially, the stance phase is the period when the foot is on the ground and the swing phase is the period when the foot is in the air. The push and impact phases are the transitions from stance to swing and from swing to stance, respectively. These four phases were selected because they are frequently used during human gait analysis [177, 178].

Figure 8.1 illustrates the four phases of gait using a simulated NAO. Figure 8.1(a) shows the push phase where the weight is transferred to the left leg and the right foot is lifted off the ground. Figure 8.1(c) shows the impact phase where the right foot regains contact with the ground and the weight is transferred back from the left leg. The swing phase is the period between the push and impact phases as shown in Figure 8.1(b), and the stance phase is the period between the impact and push phases as shown in Figure 8.1(d).

In the literature on human gait phase measurement several different types of sensors are often employed, including sensors to measure ground reaction forces and the orientation of the foot [179]. In contrast to humans, robots typically walk in such a way that both feet are parallel to the ground at all times, hence the foot orientation is approximately constant. Thus, here we use only the ground reaction forces of each foot, measured using the pressure sensors on the soles of the NAO, to determine the gait phase in real–time.

The simple state machine shown in Figure 8.2 is used to observe the weight transitions between each foot. Essentially, the ratio of force between each foot determines the current gait phase. A separate state machine is used for each leg.

Gait–phase dependent stiffness can also be applied using phase information from the walk engine. A walk engine typically has an open–loop gait phase signal used to generate the motion patterns required for walking. However, for OEM and closed source walk engines this information is not always accessible,



(a) Push (b) Swing (c) Impact (d) Stance

Figure 8.1: An image sequence of a walking simulated NAO illustrating the four phases of gait.

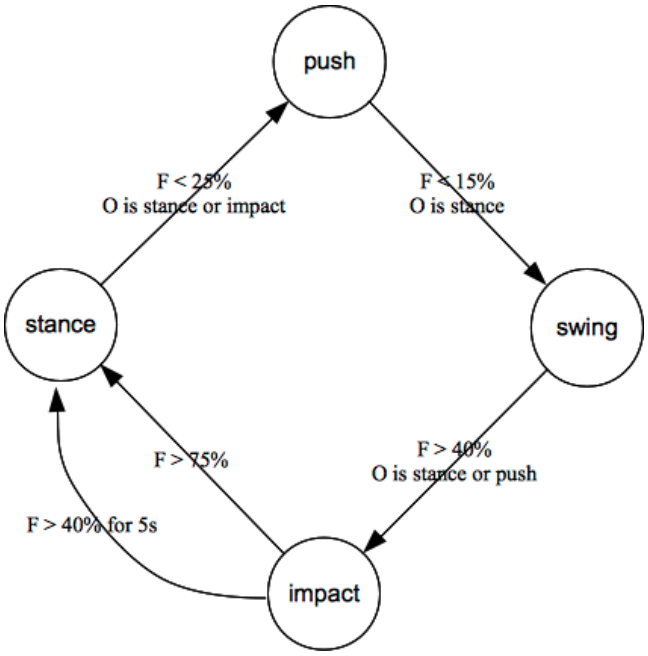


Figure 8.2: The state machine used to measure the gait phase. F is the force on the leg and O is the gait phase of the other leg.

which is the case for the NAO. Furthermore, the measurement of the gait phase if preferred because it makes the system more robust. The feet may come into contact with the ground earlier or later than expected, according to the internal phase signal, due to small trips and stumbles, or external perturbations.

Recall that in Chapters 6 and 7 the traditional walk parameter space was defined as

$$\mathcal{W}_p = \{p_1, p_2, \ldots, p_N\}$$

where N is specified by the walk engine, and the joint stiffness parameter space was defined as

$$\mathcal{W}_s = \{s_1, s_2, \ldots, s_M\}$$

where M is the number of joints.

In this chapter the joint stiffness parameter space is expanded to become gait–phase dependent via the following modification

$$\mathcal{W}_g = \{\{g_1, \ldots, g_L\}_1, \{g_1, \ldots, g_L\}_2, \ldots, \{g_1, \ldots, g_L\}_M\}$$

where M is the number of joints and L is the number of phases of gait. In this application $M = 6$ and $L = 4$, increasing the total number of stiffness parameters to 24.

The parameter space for the first application of gait–phase dependent stiffness in Section 8.2 is $\mathcal{W}_g$. Whereas, the second application in Section 8.3 makes use of the parameter space $\mathcal{W}_p \cup \mathcal{W}_g$, simultaneously optimising both the traditional parameters and the gait–phase dependent joint stiffnesses.

8.2 Phase Dependent Stiffness with Fixed Traditional Walk Parameters

8.2.1 Equipment and Method

The investigation of gait–phase dependent joint stiffnesses on a walk engine with fixed traditional walk parameters was performed on a 2009 NAO (v3.0) robot. The walk engine available for this version of NAO was very similar to that described in Section 5.2.1, however, the engine was modified to provide pseudo–omnidirectional capabilities and the ability to change direction without

stopping. Additionally, the foot sensors were used to reset the gait phase each time the feet came into contact with the ground. The reader is referred to the NUbot's team report for a description of the modifications [180], as well as the source and documentation [181] for the software written to undertake the work in this section.

The modifications to the walk engine to provide pseudo–omnidirectional walking [180] removed the ability to vary the traditional walk parameters. However, the joint stiffnesses remained real–time tuneable, thus the available walk parameter space for optimisation was $\mathcal{W}_g$. The fixed traditional walk parameters used throughout this section were those reported in Chapter 5. The initial joint stiffness parameters were also those reported in Chapter 5 where the joint stiffnesses were initially the same for each gait phase.

As previously discussed the research presented in this section was performed prior to the development of the optimisation system presented in Chapters 6 and 7. Thus, the optimisation system used in this section differs significantly. Firstly, the optimisation algorithm used in this section is the EHCLS described in Section 6.3. Secondly, the only available fitness function was the average cost of transport measured over a 20s period of forward walking. Finally, a laser–scanner based robot tracking system was implemented to provide the robot with an accurate estimate of its location and orientation.

The implementation of the laser–scanner tracking system required a significant amount of development in order to obtain sufficient accuracy. Furthermore, the system represents a novel method of tracking a humanoid robot without additional attachments. Consequently, the system is briefly reported.

Robot Position Measurement

The position of the robot is determined through edge detection. Pairs of adjacent far–near and near–far edges, that are less than the maximum size of the robot, form candidate robots. Figure 8.3 shows the edges as purple (far–near) and yellow (near–far) dots. When the system is initialised, the candidate robot closest to the centre is selected to be the actual robot. Following initialisation, a nearest neighbour approach is used to select the actual robot from the candidate edge pairs. The centroid of the data between the two edges is used as the position of the robot.

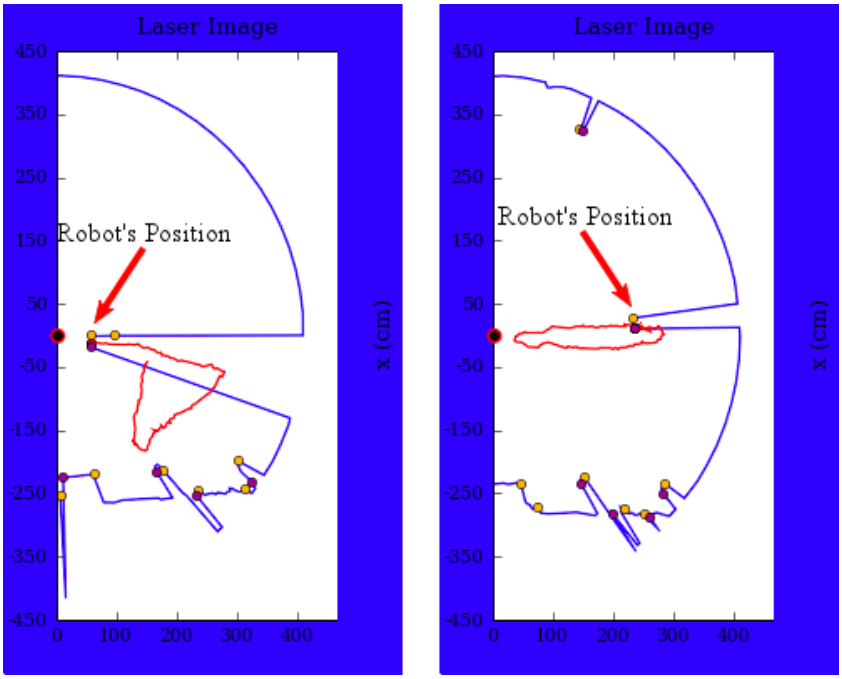


Figure 8.3: Screenshots of the tracking system showing the detected position of the robot. The laser–scanner is at the origin and the blue line represents the distances captured by the the laser–scanner. The yellow and purple markers indicate the position of the detected edges. The red line is the path travelled by the robot over the last 20s.

Several heuristics are applied to make the tracking system more robust. As the humanoid robot consists of arms and a torso it may appear as several edges in the laser scanner image. In this case the cluster of edges are merged into a single robot candidate. When the robot falls, it disappears from the laser scanner image. To ensure the system does not jump to the nearest candidate robot in the image, constraints are placed on the maximum velocity of the robot.

Robot Orientation Measurement

Figure 8.4 shows an example of the two different methods employed to measure the orientation of the robot. The panel on the left is an example of the robot being predominantly front–on to the laser–scanner. In this instance, the robot

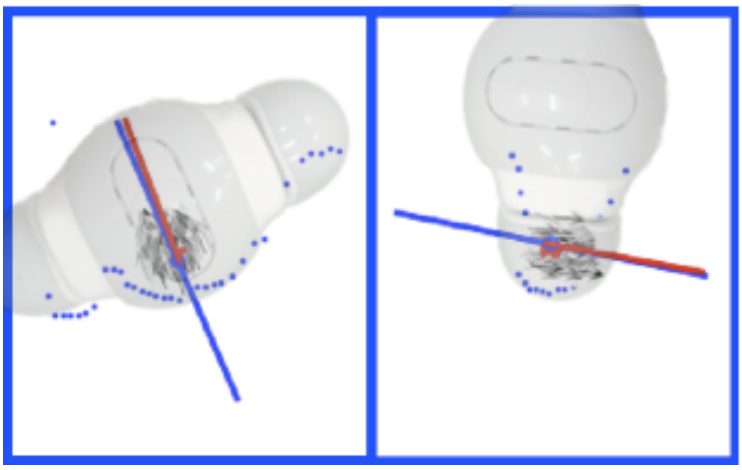


Figure 8.4: Screenshots of the tracking system showing the measured orientation of the robot. The blue markers represent the sequence of distances measured by the laser scanner, the blue line indicates the ambiguous orientation measurement, and the red line indicates the orientation estimate provided by the particle filter. For comparison the true robot orientation is indicated by the shaded regions.

image is sufficiently 'wide' and the normal to a linear least squares fit is used as a measurement for the robot's orientation. As the NAO looks the same from the front and the back, when viewed through a laser scanner, only an ambiguous measurement of the orientation is possible. This measurement is shown with the blue line in Figure 8.4, indicating that the robot could be either facing toward or away from the scanner.

The right panel of Figure 8.4 shows an example of an image captured when the robot is side–on to the scanner. In this instance, the robot image is 'narrow'. If we proceeded to fit a line to the data, we would erroneously predict that the robot was facing towards or away from the scanner. Hence, we use a threshold on the size of the image to detect that the robot image is 'narrow' and that the robot is side–on to the scanner. When a narrow image is detected, the line normal to the line between the robot's centroid and the scanner, is used as the measurement for the orientation. Again the measurement is ambiguous being modulo π.

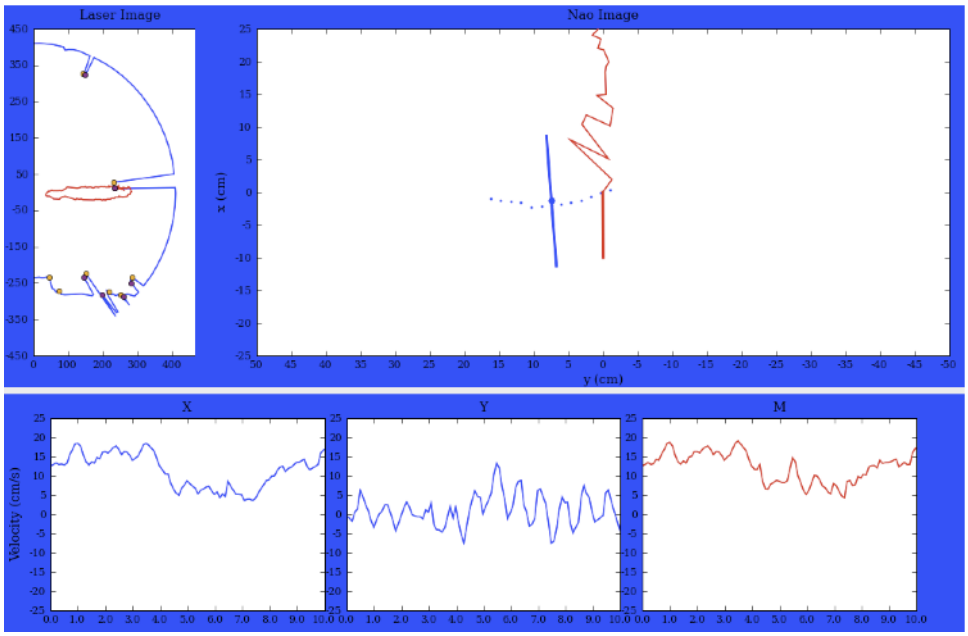


Figure 8.5: A screen shot of the robot tracking and control software. The upper left panel shows the current laser image of the environment along with the history of the robot's path. The upper right panel shows the current measurement of the robot's position and orientation, along with the current estimate from the particle filter. The lower three panels show the last ten seconds of the the measured velocities in the x, y and forward directions.

Robot Localisation

The complete robot tracking tool is shown in Figure 8.5. A Sampling Importance Resampling (SIR) particle filter [182] is used to both smooth the position estimate, improving the velocity measurement required for optimisation, and to resolve the ambiguity in the orientation measurements of the robot. The estimated state of the robot includes the velocities as they are used to measure performance, as well as to resolve the orientation ambiguity; $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$.

The certainty in the measurement for the position, orientation, and velocity is estimated based on the input image. This certainty is then used in the particle filter to appropriately weight the particles. For the position, the uncertainty is proportional to the distance from the scanner, where as the

uncertainty in the orientation is inversely proportional to the 'width' of the robot's image.

The prediction step of the particle filter has two parts. The first part estimates the new states of each particle based on their previous state using the equations:

$$\begin{aligned}
\dot{x} &= \dot{x} + \mathcal{N}(0, \sigma_{\dot{x}}) \\
\dot{y} &= \dot{y} + \mathcal{N}(0, \sigma_{\dot{y}}) \\
\dot{\theta} &= \dot{\theta} + \mathcal{N}(0, \sigma_{\dot{\theta}}/2)
\end{aligned}$$

$$\begin{aligned}
x' &= x + \dot{x}dt\cos(\dot{\theta}dt) - \dot{y}dt\sin(\dot{\theta}dt) \\
y' &= y + \dot{y}dt\cos(\dot{\theta}dt) + \dot{x}dt\sin(\dot{\theta}dt) \\
\theta' &= \theta + \dot{\theta}dt \\
\dot{x}' &= \dot{x}\cos(\dot{\theta}dt) - \dot{y}\sin(\dot{\theta}dt) \\
\dot{y}' &= \dot{y}\cos(\dot{\theta}dt) + \dot{x}\sin(\dot{\theta}dt) \\
\dot{\theta}' &= \dot{\theta} + \mathcal{N}(0, \sigma_{\dot{\theta}}/2),
\end{aligned}$$

where $\sigma_{\dot{x},\dot{y},\dot{\theta}}$ are the estimated uncertainties in the velocities, $\mathcal{N}(c, \sigma)$ is a random value from a normal distribution with mean c and variance σ, and dt is the time since the previous prediction.

The second part of the prediction step takes advantage of the fact that the control for the robot is known to the particle filter. The expected effect of the control is estimated to be a constant acceleration over a one second interval after the initial issue of the command. This constant acceleration is applied to each particle's velocity states. A one second time interval is used because the robot has a gait cycle of approximately one second and on average it takes approximately this time for the robot to start moving in the commanded direction. This second part to the prediction step allows the resolution of the ambiguous orientation measurement. For example, consider the case where the measured orientation of the robot is either facing away from or toward the laser scanner and the control commands the robot to walk forward. We can compare the measured velocity of the robot with the expected velocity to determine which way the robot is actually orientated.

When the system is initialised, particles facing in both directions are created based on an initial measurement. The states and weights of the particles are not updated until the robot starts moving. The system is hard–coded to command that the robot walk forward for the first 2 seconds, in which time, the particle filter locks onto the correct orientation.

8.2.2 Results

Recall that the walk parameters from Chapter 5 had a forward speed of 13.9cm/s and a cost of transport of 5.8J/Nm. The modifications to the walk engine to provide omnidirectional walking and foot pressure based gait phase resetting mechanism improved the speed to 14.5cm/s, but increased the cost of transport to 6.2J/N·m. It is this gait that was used as the starting point for the optimisation routine.

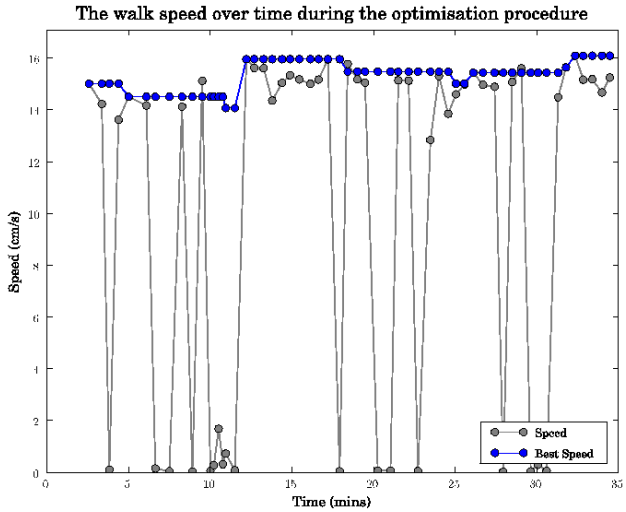
A selection of the evolution of the speed and the cost of transport during the optimisation procedure is shown in Figure 8.6. The figure spans only the first 35 minutes of the optimisation process, 35 minutes being the length of time before the battery is discharged. The figure shows that the walk speed improves to 16.1cm/s and the energy consumption reduces to 5.3J/Nm. The optimisation process is continued after the battery is replaced to achieve a final walk speed of 16.25cm/s while the efficiency remained at 5.3J/Nm. This represents an improvement in speed of 12% and an improvement in efficiency of 15% when compared to the initial walk parameters.

The particular joint stiffnesses selected by the optimisation procedure are shown in Table 8.1. The initial joint stiffnesses are also shown for comparison.

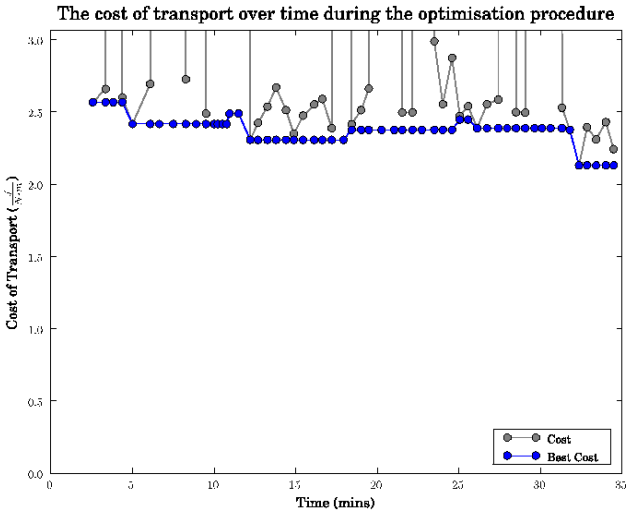
8.3 Optimisation of Phase Dependent Stiffness and Traditional Walk Parameters

8.3.1 Equipment and Method

The equipment and optimisation process used in this section are identical to that described in Chapter 6. That is, the optimisation was performed over the compact omnidirectional path shown in Figure 6.2 with the expense, $\mathcal{E}$, used in place of an iteration count. The PGRL algorithm with redundant fitness



(a) Speed



(b) Cost of transport.

Figure 8.6: The improvement in speed and cost of transport over the first 35 minutes of the optimisation procedure. The grey lines indicate the speed and efficiency of each walk tested and the blue line represents the current best values including the effect of the EHCLS resetting phase.

Table 8.1: Joint Stiffnesses Before and After Optimisation

(a) Before Optimisation

Phase	Joint Stiffness						
	HipYaw	HipRoll	HipPitch	Knee	AnklePitch	AnkleRoll	Average
All	0.70	0.26	0.55	0.25	0.24	0.28	0.38

(b) After Optimisation

Phase	Joint Stiffness						
	HipYaw	HipRoll	HipPitch	Knee	AnklePitch	AnkleRoll	Average
Stance	1.00	0.15	0.70	0.49	0.26	0.36	0.49
Push	0.59	0.06	0.30	0.22	0.27	0.16	0.27
Swing	0.56	0.19	0.45	0.14	0.37	0.22	0.32
Impact	1.00	0.29	0.50	0.26	0.24	0.45	0.46
Average	0.79	0.18	0.49	0.28	0.29	0.30	0.38

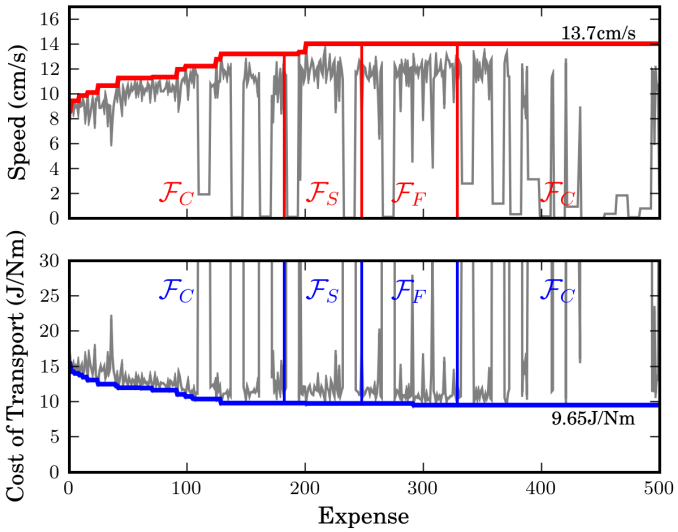
functions is used from Chapter 7, including the meta–optimised parameters and fitness functions from Chapter 6.

Here we apply the phase dependent stiffness to the 2010 NAO (v3.2) which has a significantly improved walk engine as compared to Section 8.2. In particular, it is capable of omnidirectional walking and allows the simultaneous adjustment of both traditional walk parameters and stiffness values.

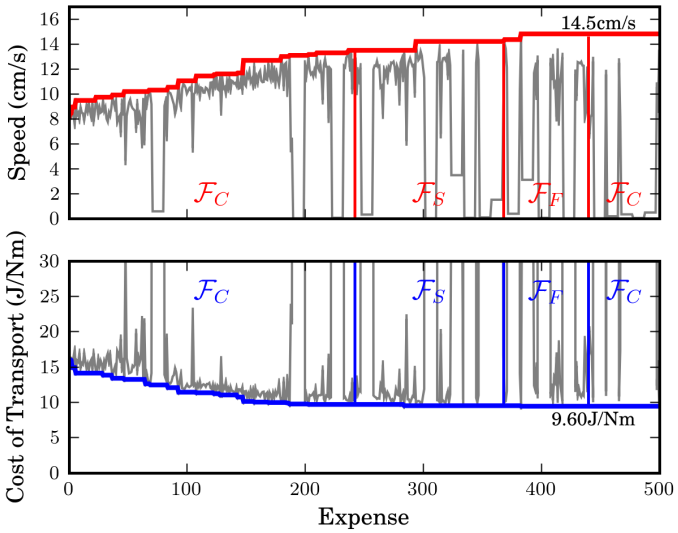
8.3.2 Results

The results of the application to the NAO are shown in Figure 8.7. Figure 8.7(a) shows the application of the optimiser on a parameter space with joint stiffness fixed over the gait cycle ($\mathcal{W}_p \cup \mathcal{W}_s$), while Figure 8.7(b) shows the results where the stiffness is gait phase dependent ($\mathcal{W}_p \cup \mathcal{W}_g$). Note that the performance of the final optimised walk in Figure 8.7(a) is significantly better than that in Figure 7.4(b), despite the optimiser and parameter space being identical in both cases. The large increase in performance between the two trials is due to the use of a previously unused NAO robot in this section.

It is clear from Figure 8.7 that the addition of the gait–phase dependent joint stiffness has improved the optimised walk. The speed of the final optimised walk improved from 13.7cm/s to 14.5cm/s, an increase of 6%, when the gait–phase dependent stiffnesses were used in place of phase independent values. The efficiency was approximately 9.6J/Nm in both cases.



(a) Fixed stiffness throughout gait cycle



(b) Variable stiffness depending on gait phase

Figure 8.7: The thick lines are the best speed and efficiency as a function of expense. The thin lines are the values for each individual trial.

8.4 Discussion

8.4.1 Phase Dependent Stiffness with Fixed Traditional Walk Parameters

Table 8.1(b) shows that the optimised stiffnesses vary significantly over the gait phases for all of the joints. Thus, the additional stiffness parameters appear to be useful in improving the walk. Furthermore, comparing Table 8.1(a) and Table 8.1(b), it is evident that the average values of the optimised phase dependent values are very similar to the initial values. That is, the average value over the gait phases for each joint is approximately equal to the global stiffness selected previously for each joint. This is particularly true for the average overall stiffness which is the same for both phase–fixed and phase–dependent sets of parameters.

The stiffness values for the stance and push phases changed the most, with an increase and decrease of 20% respectively. It is intuitive that high stiffnesses would be required during stance because the leg needs to support the weight of the robot. The significant reduction of joint stiffnesses in the push phase indicates that there is little capacity for this phase to contribute to the forward motion of the robot. This stems from the inherit trait of the walk engine where it attempts to keep the foot approximately parallel to the floor, consequently there is no scope for an actual 'push'.

There was also a significant reduction in joint stiffness in the swing phase. The swing leg is not loaded and therefore does not require as much torque to follow the target trajectories. However, it should be noted that the hip and ankle pitch joints required a high stiffness. The hip pitch joint requires a large stiffness as it moves a significant distance in the swing phase, and the walk speed is detrimentally affected if the joint fails to move this distance. The ankle pitch joint requires a high stiffness to prevent the toes from dropping and touching the ground, which can cause the robot to trip and stumble, affecting both the speed and stability.

The optimised joint stiffnesses for both the stance and impact phases of the gait had significant increases. Again this is intuitive because in these phases the leg is supporting the weight of the robot and larger torques are required. Furthermore, the significant increase, as compared to the initial

values, suggests that much of the forward movement of the walk comes from the supporting leg. The significant increase in the ankle roll stiffness in these two phases also suggests that the supporting leg does most of the work when transferring the weight between feet.

The hip yaw–pitch joint received the highest stiffness after the optimisation procedure. In fact, in the stance and impact phases, it received the maximum possible stiffness. In the NAO this joint moves both legs, hence it follows that the joint should require a higher stiffness. Furthermore, this particular joint has the most backlash in the NAO, and a high stiffness might be attempting to compensate for that.

The hip roll joints had a 70% reduction in joint stiffness after optimisation. This suggests that for this walk engine and robot it is much more efficient to transfer weight between the feet as a single inverted pendulum, that is to have very little movement of the hips.

8.4.2 Phase Dependent Stiffness with Variable Traditional Walk Parameters

The addition of the gait–phase dependent stiffness to the parameter space of the meta–optimised and improved PGRL algorithm from Chapter 7 forms the final walk optimisation system of this thesis. The average walk speed over the complex path was improved to 14.5cm/s, with peak forward and sideward speeds in excess of 20cm/s and 12cm/s, respectively. These speeds are achieved despite the limited sensor feedback used by the walk engine and the small number of available walk parameters.

The overall improvement, as compared to the default walk parameters without additional stiffness parameters, was 82% and 43% in speed and efficiency, respectively. These improvements are quite large, especially considering the starting point for the optimisation was the presumably partially optimised walk parameters provided by Aldebaran. Furthermore, the large improvement demonstrates the effectiveness of the proposed walk optimisation system.

The observed improvement in Section 8.3 when the gait–phase dependent stiffness was applied to an omnidirectional walk with adjustable walk parameters was 6%. This is significantly smaller than the results achieved when only

the stiffness could be adjusted in Section 8.2. Recall, that the optimiser employed for the former case was much more advanced and consequently better able to gain improvements in performance through the selection of traditional walk parameters.

Additionally, the walk engine used when only the stiffness could be adjusted incorporated a gait phase resetting mechanism, where a new step would be started when the foot came in contact with the ground. This feedback is very important for fast bipedal walking [59, 183] and is absent from the omnidirectional walk engine used to collect the later results. Furthermore, it was observed that during the optimisation in Section 8.2, of only the joint stiffnesses, the step frequency was increased and may explain the additional improvements.

Another major difference between the results of Sections 8.2 and 8.3 is that the former adjusted the stiffnesses along only a single direction, while the latter used a path requiring omnidirectional walking. The smaller improvement for the omnidirectional case, suggests that it may be necessary to have the stiffnesses dependent on the walk direction. The literature on humans indicates this may be necessary as the stiffness depends on the walk speed [124], and omnidirectional walking is simply the superposition of different speeds of forward and sideward walking.

8.4.3 Robot Tracking

Two different types of robot tracking methods were used in Sections 8.2 and 8.3. The laser scanner method provides accurate information about the position of the robot without the need to attach a marker. However, the orientation is difficult to measure due to the shape of the robot. Precise orientation measurements are essential if the robot is to accurately follow a desired path. The limited information provided by the laser scanner system resulted in the use of the semi–random optimisation path shown on the left in Figure 8.3. Furthermore, it meant optimisation of omnidirectional walking was not possible.

The overhead ssl–vision tracking requires the attachment of a unique marker to the robot and an environment with consistent lighting. The unique marker enables the system to accurately determine the orientation of the robot and makes the ssl–vision tracking more suitable for walk optimisation. The system

was able to guide the robot along the complex path with near perfect accuracy. The marker was easily attached in such a way that it had no effect on the walk or getup routine.

8.5 Conclusion

In this chapter the joint stiffness was specified in terms of four different gait phases for each individual joint. These additional joint stiffness parameters resulted in the improvement of the performance of two walk engines.

When gait–phase dependent stiffness was applied to the forward walk of the NAO with fixed traditional walk parameters, an improvement in the optimised speed and efficiency of 12% and 15% was achieved, respectively. The results show that gait–phase dependent stiffness can be used to improve the performance of a walk engine even when there are no available walk parameters.

Gait phase dependent stiffness was also used to enhance the optimisation of an omnidirectional walk engine. Here its use resulted in an improvement of 6% in speed. This result indicates that gait phase dependent stiffness can also improve a walk engine when used in conjunction with the traditional walk parameters.

The combination of meta–optimised PGRL, with redundant fitness functions, and a parameter space with both traditional walk parameters and gait–phase dependent stiffness is the final walk optimisation system proposed in this thesis. The overall improvement through optimisation using the proposed techniques was an 82% increase in speed, from 8.5cm/s to 14.5cm/s, and a 43% increase in efficiency, from 16.9J/Nm to 9.6J/Nm.

Chapter 9

Conclusion

9.1 Conclusions

Humanoid robot movement is an important area of robotics research. A humanoid robot has many potential applications ranging from the replacement of humans in dangerous environments, to the assistance of hospital patients and elderly people. A vast improvement in the motion system of a humanoid robot is required to fulfil these applications.

Research on humanoid robot movement also yields a greater understanding of the human motion control system. This improved understanding has benefits in the area of biomechatronics, where for example, improved protheses could be constructed, and in the treatment of disorders effecting human movement.

The two motors skills that have been the focus of this thesis are the ability to stand and walk. Studies on human motion control have highlighted several features that are yet to be applied to humanoid robots. In particular, the proprioceptive sense appears to be the most important sense for motion control. Furthermore, not only are the joint trajectories important for walking, but the joint stiffnesses also play a vital role.

The application of these important features to humanoid robots has been one of the objectives of the work presented in this thesis. The proprioceptive sense was used exclusively to perceive and quantify impacts to a standing humanoid robot. The stiffness of the joints used during walking were optimised for a humanoid robot, both independently and in conjunction with the

traditional walk parameters. Finally, performance metrics for walks, inspired by the literature on human walking, were used for the walk optimisation of a humanoid robot.

Many of the algorithms developed for humanoid robots are only applied in simulation, or to a single physical humanoid robot; they are not developed or verified on a range of humanoid robots. An aim of this thesis was to develop general techniques that could be applied to any humanoid robot. Thus, the effectiveness and generality of the techniques proposed in this thesis have been demonstrated using several physical humanoid robots.

A key consideration for the general applicability of a walk optimiser to physical humanoid robots is the stress placed on the robot during the optimisation process. We have presented several methods to reduce the amount of wear on the robot hardware. A meta–optimisation of walk optimisation techniques was performed using a performance metric incorporating the stress placed on the robot during the procedure. A local optimisation algorithm, for which the stress placed on the robot was lowest, was extended to use redundant fitness functions to allow it to escape from local maxima, thus overcoming its primary limitation.

NUPlatform Software Architecture

The application of the methods proposed in this thesis to different humanoid robots requires a software architecture that supports multiple robot platforms. To this end, the NUPlatform software architecture was designed and implemented as part of the work presented in this thesis.

In addition to enabling a stronger multi–platform verification, the strength of the software architecture itself has been demonstrated through its usefulness outside the scope of this thesis. The implementation of such a software framework requires a significant amount of time. The process is only slightly lengthened if support for multiple platforms is considered from inception, however, its utility is greatly magnified. The software framework has served as the basis for two other projects, the first being the NUbot's entry in the RoboCup soccer competitions. The second major project was in the analysis of human gaze dynamics using physical humanoid robots as simulated pedestrians.

Proprioception–based Impact Perception

The impact perception system developed using only the proprioceptive sense is able to perceive and quantify impacts to a humanoid robot. It provides all of the information, that the literature on humans suggests is necessary, to generate a postural response. Additionally, the system is also able to emulate a sense of touch without the use of a tactile skin.

Furthermore, the results of the system appear to agree with the theories presented in the literature on humans. The system performs extremely well in localising an impact, and provides a reasonable estimate of the strength and direction of impacts.

Walk Optimisation

The walk of a humanoid robot can be improved through either better design of walk engines or refinement of walk optimisation techniques. The significant improvements in the quality of the walks obtained in this thesis suggest that the optimisation techniques are just as important as the walk engines themselves.

We found that the stress placed on a robot during the optimisation process could be significantly reduced without compromising the performance of the final optimised walk. In this regard, a local optimisation algorithm performed best as it iteratively searches from an initially stable gait. To extend a local optimiser to a hybrid optimiser, the safe redundancy concept was applied using several complementary fitness functions.

We also found that an efficiency based fitness function produced the best optimised walks. The walks were smoother and exploited the natural dynamics of the robot, making them more stable and more efficient. The walks were also faster than those selected by other metrics, even a speed based fitness. These results agree with the literature on humans, where it has been suggested that an efficiency based metric is used to select a gait.

Joint Stiffness

The addition of joint stiffness parameters to the traditional trajectory–modifying walk parameters improved the speed, efficiency and stability of optimised walks. This was particularly evident on walk engines where the traditional

walk parameters were fixed or limited. In addition to adjusting the stiffness individually for each joint, improvements were achieved by adjusting the stiffness as a function of gait phase. These results also agree with the literature on humans, where it has been observed that the stiffness of each joint is different and may vary during the gait cycle.

9.2 Future Work

There are several areas for further research on the topics investigated in this thesis. These areas include extensions of the software architecture, generation of responses to impacts, and applications to a wider range of humanoid robots.

Software

The applicability of the software architecture developed here could be improved through the use of a more widely used software framework, such as ROS. The use of the ROS framework would allow for better collaboration with other developers and would give access to a host of existing implementations of common algorithms.

Additionally, the development of a ROS based framework for RoboCup soccer would be an interesting topic and encourage much more code sharing between teams and leagues.

Standing

An obvious extension to the work presented on impact perception during standing is to use the system to generate corrective responses. The system could be used to supplement an existing control system by providing additional information in regard to the impact's location and strength. Alternatively, a stand–alone control system could be developed making use of a small set of pre–generated responses, selected based on the perceived impact. Such a system is thought to be used by humans.

The impact perception system could also be applied to other humanoid robots. This would require the collection of training data for the additional robots and retraining of the SVM and SVR models to fit the new robot.

Walking

The walk optimisation and the use of joint stiffness proposed in this thesis has been applied to several robots. However, the techniques could be applied to the better walk engines that exist in the literature for the relevant robots. In particular, for the NAO there is the highly successful B–Human walk [56] and the NAO Devils walk [54], the fastest available walk. For the DARWIN-OP there is Team DARwIn's open–source walk [184].

All of the robots considered here are quite small, the application of the developed techniques to a full–size humanoid robot could also be the topic of further work. The techniques proposed in this thesis were careful to consider the stress placed on the robot during the optimisation procedure. This consideration will become even more important on full–size humanoid robots as the potential for damage is much greater.

9.3 Summary

This thesis investigated the improvement of two key skills required by humanoid robots; standing and walking. Firstly, a cross–robot and cross–platform software framework for legged robots was developed to facilitate this investigation. An impact perception system using only proprioception was successfully applied to a physical humanoid robot. An improved optimisation technique using redundant fitness functions, a meta–optimised algorithm–fitness–space combination was developed and applied over a carefully designed walking path and stress measure. The generality of the optimisation technique was demonstrated through its successful application to several different humanoid robot platforms: a simulated NAO, a physical NAO and a DARWIN-OP.

Bibliography

[1] C. Breazeal, "Emotion and sociable humanoid robots," *International Journal of Human-Computer Studies*, vol. 59, no. 1, pp. 119 – 155, 2003.

[2] P. J. Hinds, T. L. Roberts, and H. Jones, "Whose job is it anyway? a study of human-robot interaction in a collaborative task," *Human-Computer Interaction*, vol. 19, no. 1, pp. 151–181, 2004.

[3] C. Willis. World's greatest android projects. [Online]. Available: http://www.androidworld.com/prod01.htm

[4] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, "The intelligent asimo: System overview and integration," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2002, pp. 2478–2483.

[5] K. Kaneko, F. Kanehiro, M. Morisawa, K. Miura, S. Nakaoka, and S. Kajita, "Cybernetic human hrp-4c," in *IEEE-RAS Int. Conf. on Humanoid Robots*, 2009, pp. 7 –14.

[6] E. Broadbent, R. Stafford, and B. MacDonald, "Acceptance of healthcare robots for the older population: Review and future directions," *International Journal of Social Robotics*, vol. 1, pp. 319–330, 2009.

[7] L. Dickstein-Fischer, E. Alexander, X. Yan, H. Su, K. Harrington, and G. Fischer, "An affordable compact humanoid robot for autism spectrum disorder interventions in children," in *IEEE Int. Conf. on Engineering in Medicine and Biology Society*, 2011, pp. 5319 –5322.

[8] T. Ishida, Y. Kuroki, J. Yamaguchi, M. Fujita, and T. Doi, "Motion entertainment by a small humanoid robot based on open-r," in *Int. Conf. on Intelligent Robots and Systems*, vol. 2, 2001.

[9] P. Dario, E. Guglielmelli, and C. Laschi, "Humanoids and personal robots: Design and experiments," *Journal of Robotic Systems*, vol. 18, no. 12, pp. 673–690, 2001.

[10] K. Dautenhahn, S. Woods, C. Kaouri, M. Walters, K. L. Koay, and I. Werry, "What is a robot companion - friend, assistant or butler?" in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2005, pp. 1192 – 1197.

[11] H. Hasunuma, M. Kobayashi, H. Moriyama, T. Itoko, Y. Yanagihara, T. Ueno, K. Ohya, and K. Yokoil, "A tele-operated humanoid robot drives a lift truck," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 3, 2002, pp. 2246 –2252.

[12] Naval Research Laboratory, "NRL designs robot for shipboard firefighting," http://www.nrl.navy.mil/media/news-releases/2012/nrl-designs-robot-for-shipboard-firefighting, March 7 2012.

[13] W. Bluethmann, R. Ambrose, M. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder, "Robonaut: A robot designed to work with humans in space," *Autonomous Robots*, vol. 14, pp. 179–197, 2003.

[14] The Robocup Federation, "Robocup: Objective," www.robocup.org/about-robocup/objective/, 2012.

[15] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, "Robocup: A challenge problem for ai," *AI Magazine*, 1997.

[16] NASA, "Nasa spinoff magazine," spinoff.nasa.gov, 2012.

[17] R. Federation, "Robocup: Papers / publications," http://www.robocup.org/category/papers-publications/, March 2012.

[18] P. Stone, "Publications related to the robocup soccer simulation league," University of Texas, Tech. Rep., 2010.

[19] J. Kummeneje, "Robocup as a means to research, education, and dissemination, licentiate of philosophy thesis," Stockholm University, Tech. Rep., 2001.

[20] N. M. Mayer and M. Asada, "Robocup humanoid challenge," *International Journal of Humanoid Robotics*, 2008.

[21] S. Chalup, C. Murch, and M. Quinlan, "Machine learning with aibo robots in the four-legged league of robocup," *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, pp. 297–310, 2007.

[22] M. Lauer, R. Hafner, S. Lange, and M. Riedmiller, "Cognitive concepts in autonomous soccer playing robots," *Cognitive Systems Research*, vol. 11, no. 3, pp. 287 – 309, 2010.

[23] RoboCup Technical Committee, "Robocup humanoid league rules and setup," http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2011.pdf, March 2012.

[24] ——, "Robocup standard platform league rule book," http://www.tzi.de/spl/pub/Website/Downloads/Rules2011.pdf, March 2012.

[25] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *submitted to IEEE Trans. on Robotics*, 2008.

[26] I. Ha, Y. Tamura, H. Asama, J. Han, and D. W. Hong, "Development of open humanoid platform darwin-op," in *Proc. of SICE Annual Conf.*, 2011, pp. 2178 –2181.

[27] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, pp. 101–132, 2007.

[28] Tribotix: Hykim (robot bear). [Online]. Available: http://www.tribotix.com/Products/Tribotix/Robots/Hykim_info1.htm

[29] Robotis cycloidii humanoid. [Online]. Available: http://www.tribotix.com/Products/Robotis/Humanoids/CycloidII_info1.htm

[30] Aldebaran robotics' nao humanoid robot. [Online]. Available: http://www.aldebaran-robotics.com/

[31] Robotis darwin humaniod robot. [Online]. Available: sourceforge.net/projects/darwinop/

[32] S. McGill, J. Brindza, S. Yi, and D. Lee, "Unified humanoid robotics software platform," in *5th Workshop on Humanoid Soccer Robots*, 2010.

[33] [Online]. Available: http://www.naoteamhumboldt.de/en/projects/multi-platform-robot-controller/

[34] T. Pozzo, M. Ouamer, and C. Gentil, "Simulating mechanical consequences of voluntary movement upon whole-body equilibrium: the arm-raising paradigm revisited," *Biological Cybernetics*, vol. 85, pp. 39–49, 2001.

[35] L. M. Nashner, "Fixed patterns of rapid postural responses among leg muscles during stance," *Experimental Brain Research*, vol. 30, pp. 13–24, 1977.

[36] D. Wolfe and D. Winter, "Electromyographic, kinematic and kinetic responses to a perturbation of the trunk in normal standing," in *Proc. CSB Conf.*, 1987.

[37] S. Rietdyk, A. E. Patla, D. A. Winter, M. G. Ishac, and C. E. Little, "Balance recovery from medio-lateral perturbations of the upper body during standing," *J. Biomech.*, vol. 32, no. 11, pp. 1149–1158, 1999.

[38] J. S. Frank and M. Earl, "Coordination of posture and movement," *Physical Therapy*, vol. 70, no. 12, pp. 855–863, 1990.

[39] M. Vukobratović and B. Borovac, "Zero moment point – thirty five years of its life," *Int. J. Hum. Robot.*, vol. 1, no. 1, pp. 157–173, 2004.

[40] P. Sardain and G. Bessonnet, "Forces acting on a biped robot. center of pressure-zero moment point," *IEEE Trans. on Systems, Man and Cybernetics, Part A*, vol. 34, no. 5, pp. 630–637, 2004.

[41] M. Abdallah and A. Goswami, "A biomechanically motivated two-phase strategy for biped upright balance control," in *Proc. IEEE Int. Conf. on Robot. Autom.*, April 2005, pp. 1996–2001.

[42] Napoleon, S. Nakaura, and M. Sampei, "Balance control analysis of humanoid robot based on zmp feedback control," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, vol. 3, 2002, pp. 2437–2442.

[43] A. Goswami, "Postural stability of biped robots and the foot-rotation indicator (fri) point," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 523–533, 1999.

[44] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa, "Dynamics and balance of a humanoid robot during manipulation tasks," *IEEE Trans. on Robotics*, vol. 22, no. 3, pp. 568 – 575, 2006.

[45] S.-k. Yun and A. Goswami, "Momentum-based reactive stepping controller on level and non-level ground for humanoid robot push recovery," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011, pp. 3943 –3950.

[46] J. Rebula, F. Canas, J. Pratt, and A. Goswami, "Learning capture points for humanoid push recovery," in *IEEE-RAS Int. Conf. on Humanoid Robots*, 2007, pp. 65 –72.

[47] W. Mao, J.-J. Kim, and J.-J. Lee, "Continuous steps toward humanoid push recovery," in *IEEE Int. Conf. on Automation and Logistics*, 2009, pp. 7 –12.

[48] S.-J. Yi, B.-T. Zhang, D. Hong, and D. Lee, "Online learning of a full body push recovery controller for omnidirectional walking," in *IEEE-RAS Int. Conf. on Humanoid Robots*, 2011, pp. 1–6.

[49] J. J. Alcaraz-Jimenez, M. Missura, H. Martnez-Barbera, , and S. Behnke, "Lateral disturbance rejection for the nao robot," in *Proceedings of RoboCup Symposium*, 2012.

[50] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *Proc. IEEE Int. Conf. on Robot. Autom.*, 1998, pp. 1321–1326.

[51] Y. Takahashi, K. Nishiwaki, S. Kagami, H. Mizoguchi, and H. Inoue, "High-speed pressure sensor grid for humanoid robot foot," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2005, pp. 3909– 3914.

[52] M. Shimojo, T. Araki, A. Ming, and M. Ishikawa, "A zmp sensor for a biped robot," in *Proc. IEEE Int. Conf. on Robot. Autom.*, May 2006, pp. 1200–1205.

[53] Q. Huang and Y. Nakamura, "Sensory reflex control for humanoid walking," *IEEE Trans. Robot. Automat.*, vol. 21, no. 5, pp. 977–984, 2005, 1552-3098.

[54] S. Czarnetzki, S. Kerner, and O. Urbann, "Observer-based dynamic walking control for biped robots," *Robot. Auton. Syst.*, vol. 57, no. 8, pp. 839 – 845, 2009.

[55] ——, "Applying dynamic walking control for biped robots," *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 69–80, 2010.

[56] C. Graf and T. Röfer, "A closed-loop 3d-lipm gait for the robocup standard platform league humanoid," in *Proc. Workshop on Humanoid Soccer Robots*, 2010.

[57] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *Int. Journal of Robotics Research*, vol. 20, no. 2, pp. 129–143, 2001.

[58] S. Behnke, "Online trajectory generation for omnidirectional biped walking," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, Orlando, Florida, 2006, pp. 1597–1603.

[59] F. Faber and S. Behnke, "Stochastic optimization of bipedal walking using gyro feedback and phase resetting," in *IEEE Int. Conf. on Humanoid Robots*, 2007, pp. 203–209.

[60] C. Niehaus, T. Rofer, and T. Laue, "Gait optimization on a humanoid robot using particle swarm optimization," in *Proc. of Second Workshop on Humanoid Soccer Robots*, 2007.

[61] D. Gouaillier, C. Collette, and C. Kilner, "Omni-directional closed-loop walk for nao," in *IEEE Int. Conf. on Humanoid Robots*, dec. 2010, pp. 448 –454.

[62] M. Hebbel, W. Nistico, and D. Fisseler, "Learning in a high dimensional space: Fast omnidirectional quadrupedal locomotion," *RoboCup 2006: Robot Soccer World Cup X*, pp. 314–321, 2006.

[63] D. Urieli, P. MacAlpine, S. Kalyanakrishnan, Y. Bentor, and P. Stone, "On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer," in *Proc. of Int. Conf. on Autonomous Agents and Multiagent Systems*, vol. 2. IFAAMAS, May 2011, pp. 769–776.

[64] T. Hemker, H. Sakamoto, M. Stelzer, and O. von Stryk, "Hardware-in-the-loop optimization of the walking speed of a humanoid robot," in *Proc. Int. Conf. on Climbing and Walking Robots*, 2006, pp. 614–623.

[65] C. Meriçli and M. Veloso, "Improving biped walk stability using real-time corrective human feedback," *RoboCup 2010: Robot Soccer World Cup XIV*, pp. 194–205, 2011.

[66] J. Zagel and J. R. del Solar, "Combining simulation and reality in evolutionary robotics," *Journal of Intelligent and Robotic Systems*, vol. 59, no. 1, pp. 19–39, 2007.

[67] D. Gong, J. Yan, and G. Zuo, "A review of gait optimization based on evolutionary computation," *Applied Computational Intelligence and Soft Computing*, 2010.

[68] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *National Conference on Artificial Intelligence*, 2004.

[69] M. Pedersen, "Good parameters for particle swarm optimization," Hvass Laboratories, Tech. Rep., 2010.

[70] M. Meissner, M. Schmuker, and G. Schneider, "Optimized particle swarm optimization (opso) and its application to artificial neural network training," *BMC Bioinformatics*, vol. 7, pp. 1–11, 2006.

[71] G. Capi, S. Kaneko, K. Mitobe, L. Barolli, and Y. Nasu, "Optimal trajectory generation for a prismatic joint biped robot using genetic algorithms," *Robotics and Autonomous Systems*, vol. 38, no. 2, pp. 119 – 128, 2002.

[72] J. H. Park and M. Choi, "Generation of an optimal gait trajectory for biped robots using a genetic algorithm," *JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing*, vol. 47, no. 2, pp. 715–721, 2004.

[73] L. Hu, C. Zhou, and Z. Sun, "Estimating probability distribution with q-learning for biped gait generation and optimization," in *Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 362 –367.

[74] M. Alamir, J. S. Welsh, and G. C. Goodwin, "Redundancy versus multiple starting points in nonlinear system related inverse problems," *Automatica*, vol. 45, pp. 1052–1057, 2009.

[75] Z. Matjacic, "Control of ankle and hip joint stiffness for arm-free standing in paraplegia," *Neuromodulation*, vol. 4, no. 1, pp. 37–46, 2001.

[76] C. Capaday and R. Stein, "Amplitude modulation of the soleus h-reflex in the human during walking and standing," *Journal of Neuroscience*, vol. 6, no. 5, pp. 1308–1313, 1986.

[77] X. Duan, R. Allen, and J. Sun, "A stiffness-varying model of human gait," *Medical Engineering & Physics*, vol. 19, no. 6, pp. 518 – 524, 1997.

[78] Newcastle robotics lab: Publications and reports. [Online]. Available: http://robots.newcastle.edu.au/publications.html

[79] J. Kulk, S. Nicklin, and A. Wong. (2011) Nubot's robocup code repository. [Online]. Available: http://github.com/nubot/robocup

[80] *Bioinspiration & Biomimetics*, 2012.

[81] R. Pfeifer, M. Lungarella, and F. Iida, "Self-organization, embodiment, and biologically inspired robotics," *Science*, 2007.

[82] Y. Bar-Cohen, *Biomimetics: Biologically Inspired Technologies*. CRC Press, 2005.

[83] [Online]. Available: http://www.bostondynamics.com/robot_cheetah.html

[84] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *Science*, 2007.

[85] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, 2002.

[86] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems*, 2003.

[87] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, vol. 307, no. 5712, pp. 1082–1085, 2005.

[88] W. McIlroy and B. Maki, "Preferred placement of the feet during quiet stance: development of a standardized foot placement for balance testing," *Clinical Biomechanics*, vol. 12, no. 1, pp. 66–70, 1997.

[89] D. Winter, "Human balance and posture control during standing and walking," *Gait & Posture*, vol. 3, no. 4, pp. 193–214, 1995.

[90] D. A. Winter, *Biomechanics and motor control of human movement*, 3rd ed. John Wiley & Sons, 2004.

[91] P. M. Kennedy and J. T. Inglis, "Distribution and behaviour of glabrous cutaneous receptors in the human foot sole," *Journal of Physiology*, vol. 538, no. 3, pp. 995–1002, 2002.

[92] P. F. Meyer, L. I. Oddsson, and C. J. D. Luca, "The role of plantar cutaneous sensation in unperturbed stance," *Experimental Brain Research*, vol. 156, pp. 505–512, 2004.

[93] H. Mittelstaedt, "Somatic graviception," *Biological Psychology*, vol. 42, pp. 53–74, 1996.

[94] ——, "Origin and processing of postural information," *Neuroscience & Biobehavioral Reviews*, vol. 22, no. 4, pp. 473–478, 1998.

[95] C. Gianna, S. Heimbrand, and M. Gresty, "Thresholds for detection of motion direction during passive lateral whole-body acceleration in normal subjects and patients with bilateral loss of labyrinthine function," *Brain Research Bulletin*, vol. 40, pp. 443–447, 1996.

[96] A. Benson, M. Spencer, and J. Stott, "Thresholds for the detection of the direction of whole-body, linear movement in the horizontal plane," *Aviat Space Environ Med.*, vol. 57, no. 11, 1986.

[97] H. Kingma, "Thresholds for perception of direction of linear acceleration as a possible evaluation of the otolith function," *BMC Ear Nose Throat Disord*, vol. 5, no. 5, 2005.

[98] L. Bringoux, S. Schmerber, V. Nougier, G. Dumas, P. A. Barraud, and C. Raphel, "Perception of slow pitch and roll body tilts in bilateral

labyrinthine-defective subjects," *Neuropsychologia*, vol. 40, no. 4, pp. 367–372, 2002.

[99] R. Fitzpatrick and D. I. McCloskey, "Proprioceptive, visual and vestibular thresholds for the perception of sway during standing in humans," *J Physiol.*, vol. 478, no. 1, pp. 173–186, 1994.

[100] W. Paulus, A. Straube, S. Krafczyk, and T. Brandt, "Differential effects of retinal target displacement, changing size and changing disparity in the control of anterior/posterior and lateral body sway," *Experimental Brain Research*, vol. 78, no. 2, pp. 243–252, 1989.

[101] L. Nashner, F. Black, and d. Wall, C, "Adaptation to altered support and visual conditions during stance: patients with vestibular deficits," *J. Neurosci.*, vol. 2, no. 5, pp. 536–544, 1982.

[102] G. Simoneau, J. Ulbrecht, J. Derr, and P. Cavanagh, "Role of somatosensory input in the control of human posture," *Gait & Posture*, vol. 3, no. 3, pp. 115–122, 1995.

[103] F. Horak, R. Dickstein, and R. Peterka, "Diabetic neuropathy and surface sway-referencing disrupt somatosensory information for postural stability in stance," *Somatosensory & Motor Research*, vol. 19, pp. 316–326, 2002.

[104] A. Fu and C. Hui-Chan, "Ankle Joint Proprioception and Postural Control in Basketball Players With Bilateral Ankle Sprains," *Am J Sports Med*, vol. 33, no. 8, pp. 1174–1182, 2005.

[105] O. Sacks, *The disembodied lady. The man who mistook his wife for a hat.* London: Picador, 1985, pp. 42–52.

[106] J. Jeka, T. Kiemel, R. Creath, F. Horak, and R. Peterka, "Controlling Human Upright Posture: Velocity Information Is More Accurate Than Position or Acceleration," *J Neurophysiol*, vol. 92, no. 4, pp. 2368–2379, 2004.

[107] I. Hunter and R. Kearney, "Dynamics of human ankle stiffness: Variation with mean ankle torque," *Journal of Biomechanics*, vol. 15, no. 10, pp. 747 – 752, 1982.

[108] I. D. Loram and M. Lakie, "Direct measurement of human ankle stiffness during quiet standing: the intrinsic mechanical stiffness is insufficient for stability," *J Physiol*, vol. 545, no. 3, pp. 1041–1053, 2002.

[109] P. G. Morasso and V. Sanguineti, "Ankle Muscle Stiffness Alone Cannot Stabilize Balance During Quiet Standing," *J Neurophysiol*, vol. 88, no. 4, pp. 2157–2162, 2002.

[110] D. A. Winter, A. E. Patla, F. Prince, M. Ishac, and K. Gielo-Perczak, "Stiffness Control of Balance in Quiet Standing," *J Neurophysiol*, vol. 80, no. 3, pp. 1211–1221, 1998.

[111] J. H. J. Allum, B. R. Bloem, M. G. Carpenter, M. Hulliger, and M. Hadders-Algra, "Proprioceptive control of posture: a review of new concepts," *Gait & Posture*, vol. 8, no. 3, pp. 214 – 242, 1998.

[112] B. R. Bloem, J. H. J. Allum, M. G. Carpenter, and F. Honegger, "Is lower leg proprioception essential for triggering human automatic postural responses?" *Experimental Brain Research*, vol. 130, no. 3, pp. 375–391, 2000.

[113] B. Bloem, J. Allum, M. Carpenter, J. Verschuuren, and F. Honegger, "Triggering of balance corrections and compensatory strategies in a patient with total leg proprioceptive loss," *Exp. Brain Res.*, vol. 142, pp. 91–107, 2002.

[114] G. Torres-Oviedo and L. H. Ting, "Muscle Synergies Characterizing Human Postural Responses," *J Neurophysiol*, vol. 98, no. 4, pp. 2144–2156, 2007.

[115] M. C. Tresch and A. Jarc, "The case for and against muscle synergies," *Current Opinion in Neurobiology*, vol. 19, no. 6, pp. 601 – 607, 2009.

[116] P. F. Meyer, L. I. Oddsson, and C. J. D. Luca, "Reduced plantar sensitivity alters postural responses to lateral perturbations of balance," *Exp. Brain Res.*, vol. 157, pp. 526–536, 2004.

[117] H. J. Ralston, "Energetics of human walking," *Neural Control of Locomotion*, 1976.

[118] K. G. Holt, S. F. Jeng, R. Ratcliffe, and J. Hamill, "Energetic cost and stability during human walking at the preferred stride frequency," *Journal of Motor Behavior*, vol. 27, no. 2, pp. 164–178, 1995.

[119] J. Maxwell Donelan, R. Kram, and K. Arthur D., "Mechanical and metabolic determinants of the preferred step width in human walking," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 268, no. 1480, pp. 1985–1992, 2001.

[120] F. C. Anderson and M. G. Pandy, "Dynamic optimization of human walking," *Journal of Biomechanical Engineering*, vol. 123, no. 5, pp. 381–390, 2001.

[121] G. Yamaguchi and F. Zajac, "Restoring unassisted natural gait to paraplegics via functional neuromuscular stimulation: a computer simulation study," *IEEE Trans. Biomedical Engineering*, vol. 37, no. 9, pp. 886 –902, 1990.

[122] L. Ren, R. K. Jones, and D. Howard, "Predictive modelling of human walking over a complete gait cycle," *Journal of Biomechanics*, vol. 40, no. 7, pp. 1567 – 1574, 2007.

[123] R. B. Davis and P. A. DeLuca, "Gait characterization via dynamic joint stiffness," *Gait & Posture*, vol. 4, no. 3, pp. 224 – 231, 1996.

[124] A. H. Hansen, D. S. Childress, S. C. Miff, S. A. Gard, and K. P. Mesplay, "The human ankle during walking: implications for design of biomimetic ankle prostheses," *Journal of Biomechanics*, vol. 37, no. 10, pp. 1467 – 1474, 2004.

[125] A. S. W. Wong, S. K. Chalup, S. Bhatia, A. Jalalian, J. Kulk, and M. J. Ostwald, "Humanoid robots for modelling and analysing visual gaze dynamics of pedestrians moving in urban space," in *The 45th Annual Conf. of the Australian and New Zealand Architectural Science Association (ANZASCA2011)*, 2011.

[126] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *The Int. Journal of Robotics Research*, vol. 17, no. 4, p. 315, 1998.

[127] V. Jagannathan, R. Dodhiawala, and L. Baum, *Blackboard architectures and applications*. Boston: Academic Press, 1989, vol. 3.

[128] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[129] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[130] T. Röfer, T. Laue, J. Müller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. de Haas *et al.*, "B-human team report and code release 2010," in *Technical report. http://www. b-human. de/en/publications*, 2010.

[131] S. Barrett, K. Genter, M. Hausknecht, T. Hester, P. Khandelwal, J. Lee, M. Quinlan, A. Tian, P. Stone, and M. Sridharan, "Austin villa 2010 standard platform team report," Technical Report UT-AI-TR-11-01, The

University of Texas at Austin, Department of Computer Sciences, AI Laboratory, Tech. Rep., 2011.

[132] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework fawkes," in *Simulation, Modeling, and Programming for Autonomous Robots*, 2010, pp. 300–311.

[133] T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. of the Australasian Conf. on Robotics and Automation (ACRA 2005)*, 2005.

[134] S. Petters, D. Thomas, and O. Von Stryk, "Roboframe-a modular software framework for lightweight autonomous robots," in *Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007.

[135] (2010) Northern bites' robocup code repository. [Online]. Available: http://github.com/northern-bites/nao-man

[136] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[137] Robocup gamecontroller. [Online]. Available: http://sourceforge.net/projects/robocupgc/

[138] Kitware. Cmake: Cross platform make.

[139] (2010) Cyberbotics webots. [Online]. Available: http://www.cyberbotics.com/products/webots/

[140] S. P. Nicklin, S. Bhatia, D. Budden, R. A. King, J. Kulk, J. Walker, A. S. Wong, and S. K. Chalup, "The nubots' team description for 2011," University of Newcastle, Tech. Rep., 2011. [Online]. Available: http://www.tzi.de/spl/pub/Website/Teams2011/NUbotsTDP2011.pdf

[141] M. Quinlan, C. Murch, R. Middleton, and S. Chalup, "Traction monitoring for collision detection with legged robots," in *Proceedings of RoboCup Symposium*, 2003.

[142] C. Stanton, E. Ratanasena, S. Haider, and M.-A. Williams, "Perceiving forces, bumps, and touches from proprioceptive expectations," in *Proceedings of RoboCup Symposium*, 2011, pp. 301 – 312.

[143] J. Hoffmann and D. Göhring, "Sensor-actuator-comparison as a basis for collision de- tection for a quadruped robot," in *Proceedings of RoboCup Symposium*, 2004.

[144] M. J. Quinlan, "Machine learning on aibo robots," Ph.D. dissertation, School of Electrical Engineering and Computer Science, University of Newcastle, Australia, 2006.

[145] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[146] J. Painter, D. Kerstetter, and S. Jowers, "Reconciling steady-state kalman and alpha-beta filter design," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 6, pp. 986 –991, nov 1990.

[147] R. Krohling, "Gaussian swarm: a novel particle swarm optimization algorithm," in *Proc. of IEEE Conf. on Cybernetics and Intelligent Systems*, vol. 1, 2004, pp. 372 – 376.

[148] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of NAO humanoid," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2009, pp. 769–774.

[149] Robotis darwin humaniod robot. [Online]. Available: http://www.robotis.com/xe/darwin_en

[150] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, Mar 2002.

[151] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 09 1995.

[152] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[153] A. Scheiner, D. Ferencz, and H. Chizeck, "The effect of joint stiffness on simulation of the complete gait cycle," in *Proc. Int. Conf. Engineering Advances: New Opportunities for Biomedical Engineers*, 1994, pp. 386 –387.

[154] H.-o. Lim, S. Setiawan, and A. Takanishi, "Balance and impedance control for biped humanoid robot locomotion," in *Proc.s of IEEE Int.l Conf. on Intelligent Robots and Systems*, 2001, pp. 494–499.

[155] S. Kawaji, K. Ogasawara, J. Iimori, and S. Yamada, "Compliance control for biped locomotion robot," in *Proc. of IEEE Int. Conf. on Systems, Man, and Cybernetics*, 1997, pp. 3801–3806.

[156] N. Nishikawa, Y. Fujimoto, T. Murakami, and K. Ohnishi, "Variable compliance control for 3 dimensional biped robot considering environmental fluctuations," *Trans. of the Institute of Electrical Engineers of Japan*, vol. 119, no. 12, pp. 1507–1514, 1999.

[157] S. Sakaino and K. Ohnishi, "Sliding mode control based on position control for contact motion applied to hopping robot," in *Proc. of IEEE Int. Conf. on Industrial Technology*, 2006, pp. 170–175.

[158] M. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, 1964.

[159] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 3, apr. 2004, pp. 2619 – 2624.

[160] J. Strom, G. Slavov, and E. Chown, "Omnidirectional walking using zmp and preview control for the nao humanoid robot," *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 378–389, 2010.

[161] N. Shafii, L. Reis, and N. Lau, "Biped walking using coronal and sagittal movements based on truncated fourier series," *RoboCup 2010: Robot Soccer World Cup XIV*, pp. 324–335, 2011.

[162] A. Cherubini, F. Giannone, L. Iocchi, M. Lombardo, and G. Oriolo, "Policy gradient learning for a humanoid soccer robot," *Robotics and Autonomous Systems*, vol. 57, no. 8, pp. 808 – 818, 2009.

[163] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone, "Design and optimization of an omnidirectional humanoid walk:a winning approach at the robocup 2011 3d simulation competition," in *AAAI Conference on Artificial Intelligence*, 2012.

[164] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso, "Ssl-vision: The shared vision system for the robocup small size league," *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 425–436, 2010.

[165] M. J. Quinlan, S. K. Chalup, and R. H. Middleton, "Techniques for improving vision and locomotion on the sony aibo robot," in *Proc. of Australasian Conf. on Robotics and Automation*, 2003.

[166] C. L. Vaughan and M. J. O'Malley, "Froude and the contribution of naval architecture to our understanding of bipedal locomotion," *Gait & posture*, vol. 21, no. 3, pp. 350–362, 2005.

[167] (2010) Robocup 2010: Nao walking at 444 mm/s. [Online]. Available: http://www.youtube.com/watch?v=3aOuQ1_e--k

[168] (2010) B-human - runswift (robocup 2010, spl, final). [Online]. Available: http://www.youtube.com/watch?v=z5-11Awt0Rw

[169] M. Alamir, "On useful redundancy in dynamic inverse problems related optimization," in *IFAC World Congress*, 2008.

[170] H. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," in *Int. Conf. on Computational Intelligence for Modelling, Control and Automation*, vol. 1, 2005, pp. 695 –701.

[171] S. Rahnamayan, H. Tizhoosh, and M. Salama, "Opposition-based differential evolution," *IEEE Trans. on Evolutionary Computation*, vol. 12, no. 1, pp. 64 –79, 2008.

[172] (2011) Darwin-op downloads page. [Online]. Available: http://sourceforge.net/projects/darwinop/

[173] D. D. L. S.-J. Y. S. M. Y. Z. S. B. M. M. H. S. D. Hong, J. Han, and M. Hopkins. (2011) Robocup 2011 humanoid league winners. [Online]. Available: http://www1.cs.columbia.edu/~allen/F11/NOTES/RoboCup.pdf

[174] J.-Y. Kim, C. Atkeson, J. Hodgins, D. Bentivegna, and S. J. Cho, "Online gain switching algorithm for joint position control of a hydraulic humanoid robot," in *Proc. Int. Conf. Humanoid Robots*, 2007.

[175] J. Roberts, D. Kee, and G. Wyeth, "Improved joint control using a genetic algorithm for a humanoid robot," in *Proc. Australasian Conference on Robotics and Automation*, 2003.

[176] D. K. L. Kee, G. F. Wyeth, and J. Roberts, "Biologically inspired joint control for a humanoid robot," in *Proc. Int. Conf. Humanoid Robots*, 2004.

[177] E. P. Zehr and R. B. Stein, "What functions do reflexes serve during human locomotion?" *Progress in Neurobiology*, vol. 58, pp. 185–205, 1999.

[178] C. L. Vaughan, B. L. Davis, and J. C. O'Connor, *Dynamics of Human Gait*, 2nd ed., C. L. Vaughan, Ed. Howard Place, Western Cape 7450, South Africa: Kiboho Publishers, 1999.

[179] I. Pappas, M. Popovic, T. Keller, V. Dietz, and M. Morari, "A reliable gait phase detection system," *IEEE Trans. on Neural Systems and Rehabilitation Engineering*, vol. 9, no. 2, pp. 113 –125, 2001.

[180] N. Henderson, S. P. Nicklin, A. Wong, J. Kulk, S. K. Chalup, and R. King, "The 2009 nubots team report," University of Newcastle, Australia, Tech. Rep., 2009.

[181] J. Kulk. (2009) naowalkoptimiser github source code repository. [Online]. Available: http://github.com/soneoed/naowalkoptimiser

[182] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Trans. on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[183] C. Graf, A. Hartl, T. Rofer, and T. Laue, "A robust closed-loop gait for the standard platform league humanoid," in *Proc. Workshop on Humanoid Soccer Robots*, 2009.

[184] [Online]. Available: https://fling.seas.upenn.edu/~robocup/wiki/index.php