International Master's Thesis

Model predictive control of a walking bipedal robot using online optimization

Alexander Sherikov
Technology

Studies from the Department of Technology at Örebro University
Örebro 2012

Model predictive control of a walking bipedal robot using online optimization

Studies from the Department of Technology
at Örebro University

Alexander Sherikov

Model predictive control of a walking bipedal robot using online optimization

Supervisor: **Dr. Dimitar Dimitrov**

Examiners: **Dr. Mathias Broxvall**
Dr. Marcus Sundhäll

Title: Model predictive control of a walking bipedal robot using online optimization

ISSN

Abstract

Humanoid robotics is a challenging and promising research field. Legged locomotion is one of the most important aspects of it. In spite of the progress achieved in the last years in control of walking robots, many problems are yet to be resolved. The inherent complexity of such robots makes their control a difficult task even on the modern hardware. In order to address this issue approximate models and high performance algorithms are employed. This thesis is focused on the model predictive control of a walking bipedal robot, which is approximated by an inverted pendulum, using online optimization. A special emphasis is made on the solvers that exploit the structure of quadratic optimization problems in the context of model predictive control. Two methods for solution of these problems are implemented: primal active set and primal logarithmic barrier methods. They are tested and compared in a simulation and on a humanoid robot. A software module for control of the Nao humanoid robot is developed for this purpose.

Acknowledgements

First of all I want to thank my parents and sister, without their support and understanding this work would be impossible. Also, I would like to thank my supervisor Dimitar Dimitrov, whose guidance and advice were indispensable.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3D-LIPM	Three-dimensional Linear Inverted Pendulum Model
API	Application Programming Interface
CoM	Center of Mass
CoP	Center of Pressure
DCM	Device Communication Manager
DS	Double Support
FK	Forward Kinematics
FZMP	Fictitious Zero Moment Point
IGM	Inverse Geometrical Model
IK	Inverse Kinematics
KKT	Karush-Kuhn-Tucker conditions
LMPC	Linear Model Predictive Control
MPC	Model Predictive Control
MPCWMG	Model Predictive Control for Walking Motion Generation
NMPC	Nonlinear Model Predictive Control
PoS	Polygon of Support
QP	Quadratic Program
SDK	Software Development Kit
SS	Single Support
ZMP	Zero Moment Point

Chapter 1
Introduction

1.1 Motivation

Creation of artificial humans has been a prominent idea in many cultures across the world [20]. However, even replication of the simplest functions of human brain and body is difficult, if possible, with the current development of technologies. Humanoid robotics tries to develop robots that are capable of working in the environments adapted for humans, side by side with them or instead of them. It is no wonder, that complexity and diversity of these problems are appealing for many researchers. This thesis is focused only on one aspect of development of humanoid robots – on their locomotion.

Obviously, humanoid robots have to use their legs for locomotion. Modern research in this field begins in the 70's with the projects of I. Kato in Japan at Waseda University and M. Vukobratović at Mihalo Pupin Institute, Belgrade. The first group worked on anthropomorphic robots, while the second one – on active exoskeletons for rehabilitation. In spite of the progress made since then, many problems still do not have practical solutions. One of the most important restricting factors is complexity and intrinsic nonlinearity of models of humanoid robots. It makes the control of such robots challenging in real-time on the available computing hardware. Though explicit control is not always necessary (for example, for passive walkers, which are briefly described in Chapter 2), it usually gives more flexibility. Hence, it is often necessary to employ approximate models, as well as highly optimized software to realize motions of humanoids. The goal of the thesis is to design, implement and test on a humanoid platform an embedded optimization based control scheme for walking.

1.2 Contribution

Apart from the work of many researchers in the field of legged locomotion this thesis is a successor to the project of Antonio Paolillo [28]. The main contribution of this thesis is development of an optimized solver for the quadratic

problem (refer to Chapter 4), which is used for walking motion generation. The solver exploits the structure of the sparse formulation of the quadratic problem in order to achieve higher performance. The design of this solver and its performance are discussed in

- D. Dimitrov, A. Sherikov, and P.B. Wieber. A sparse model predictive control formulation for walking motion generation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2292–2299. IEEE, 2011.

Furthermore, the software module for control of a Nao robot (see Section 2.4) developed in [28] was rewritten and extended. The new version implements closed loop control with error feedback and uses the position of the center of mass for control decisions instead of position of the torso (refer to Chapter 5 for more information).

1.3 Outline

The thesis consists of seven chapters including this introductory chapter. Chapter 2 introduces terminology, describes basic concepts and reviews related works. The formulation of model predictive control problem for walking pattern generation and some general notes on the model predictive control are given in Chapter 3. The next Chapter 4 discusses implementation of optimized solvers for the forenamed model predictive control problem. The design of software walking module for the Nao robot and the results of the experiments on the robot and in a simulation are presented in Chapter 5 and Chapter 6, respectively. The results are summarized in Chapter 7, which also discusses possible future work. Appendix A contains sample derivations of the Schur complement, which is used to solve the model predictive control problem.

1.4 Notation

Names of programs and software libraries, names of constants, variables and functions that are used in programs are typed in monospace font, for example, `Eigen`.

Matrices are denoted by bold capital letters, vectors – by bold letters in lower case, scalars – by letters in lower case, for example, $\boldsymbol{C}, \boldsymbol{x}, y$.

Quadratic forms in mathematical expressions are denoted as

$$\|\boldsymbol{x}\|_{\boldsymbol{Q}}^2 \triangleq \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x}.$$

Chapter 2
Background

2.1 Basic Terminology

This section contains definitions of some basic terms that are used in the thesis.

Walking bipedal robots are robotic systems that can walk using two legs. Terms **biped**, which stands for any two-footed animal or robot; **humanoid**, which denotes mechanical systems or creatures having human appearance; and **walking bipedal robot** are used interchangeably in this thesis.

Walking is defined as a locomotion of a system having multiple contacts with the ground by means of breaking and regaining these contacts without simultaneous breaking of all contacts. A typical **walking cycle** of a biped is shown in Figure 2.1.

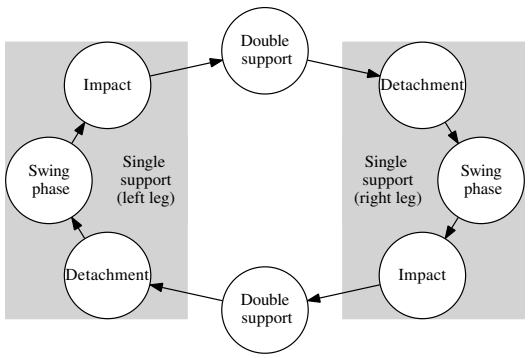


Figure 2.1: Walking cycle of a biped is typically defined in such a way, that it includes two swing phases of right and left leg and two phases when both feet are on the ground.

The convex hull of the ground contacts in the ground plane is called the **support area**. In practical applications it is often represented by a polygon, hence an equivalent term **Polygon of Support (PoS)** is often used.

The phase of the walking cycle when only one foot contact with the ground is preserved is referred to as **Single Support** (**SS**), while the phase when both feet are in contact with the ground is called **Double Support** (**DS**). **SS** and **DS** are also used to denote the respective support area.

Step is defined as a half of walking cycle, which includes one single support and adjacent double support. **Footstep** denotes the position of a foot in ground plane. The term "gait" is used to denote a pattern of movements of a robot during walk.

In accordance with [31] **balance** is defined as the state, in which humanoid preserves the upright position. Some authors use the word "stability" instead of "balance", but such use is avoided here to prevent confusion. Terms **static balance** and **dynamic balance** are used to discriminate situations when a robot is balanced while it is still, and while it is moving. The ability to preserve balance is a crucial characteristic for walking robots.

2.2 Dynamic Balance

The dynamic balance of a biped can be ensured using one of the criteria presented in this section. Note that it is possible to design a gait, which is not balanced with respect even to the most general criteria mentioned here. For example, a gait may include a phase of controlled fall, when the fall is started intentionally and appropriate preparations are made in order to continue the walk after the foot impact. Nevertheless, design of such gaits is not a trivial task.

Also, there is a class of robots, which are designed using a completely different paradigm. Continuous walk of such robots is characterized by a cycle in the phase plane, which is supported either naturally or using control [17]. **Passive walkers**, which were introduced in [23], are the best known representatives of this class of systems. Walking is a natural dynamic mode of passive walkers. Their mechanical design allows them to walk on shallow slopes without any active control. Combination of passive dynamics with actuation is also a promising research direction [6].

2.2.1 Center of Mass

One of the best known balance criteria is the position of the projection of the **Center of Mass** (**CoM**) on the ground plane, which must stay within the support area. This criterion was used in other engineering disciplines before it was adopted in robotics. Robots that balance using projection of **CoM** are called **static walkers** to indicate, that the static balance is always preserved. Hence, they can be safely stopped at any moment.

Static walkers are rather limited in their capabilities, in particular their walk is rather slow, since they must limit their acceleration. Note that this criterion is

valid only when all ground contacts of a robot lie on the same horizontal plane [33].

2.2.2 Zero Moment Point

This criterion, which is based on the existence of **Zero Moment Point** (**ZMP**) [30], was proposed by Miomir Vukobratović in 1968 at the Third All-Union Congress on Theoretical and Applied Mechanics in Moscow.

The concept of **ZMP** is introduced under the assumption that a robot walks on a flat floor and the friction forces are strong enough to compensate the ground reaction forces tangential to the ground.

Since the ground contact of a robot is unilateral (the robot can push on the ground, but cannot pull), the only force that can compensate forces that tend to overturn the robot is the ground reaction force. The point where the ground reaction force must be applied to compensate for other forces must exist within the support area, otherwise the robot would lose balance. This point is named **ZMP**.

When the **ZMP** does not exist within the support area, under the assumption that the support is immobile (which is not true in this case) a **Fictitious Zero Moment Point** (**FZMP**) can be found in order to measure the disbalance [31].

The position of **Center of Pressure** (**CoP**), which is a a point on the ground plane, where the ground reaction force is applied, is often considered to be an equivalent of **ZMP**, since they coincide while the robot is balanced. Therefore, the actual position of **ZMP** can be found using force sensors located on the soles of a robot.

Even though the situation when **ZMP** exists on the edge of the support area do not necessary lead to fall, it is dangerous since any small disturbance may overturn the robot. The easiest way to avoid this is to define support area with a safety margin.

2.2.3 Other Criteria

The forenamed criteria cannot be used on uneven terrain, but a more general criteria can be developed [33, 5, 15]. Nevertheless, they are computationally more expensive and may be infeasible for real-time applications [33].

2.3 3D Linear Inverted Pendulum Model

Apart from balancing there are other factors that complicate control of walking robots. Hence, it is common to make simplifying assumptions. For example, in this thesis it is assumed that walk is performed in such a way, that constraints imposed by environment, joint limits, self-collision avoidance are never violated.

However, it is necessary to make other assumptions. Consider generation of trajectories for a humanoid in the joint or operational space. If these trajectories are generated offline, the motion cannot adapt to the walk conditions. Hence, it is more appealing to generate motion profiles online. This can be achieved with the help of **Model Predictive Control** (**MPC**) [17], refer to Chapter 3 for more information on **MPC**. If the whole model of a robot robot is incorporated in the **MPC**, then the respective optimization problem becomes nonlinear [17]. Real-time control based on **Nonlinear Model Predictive Control** (**NMPC**) is not always feasible, since its application is a computationally expensive task. This limitation motivated researchers to develop control schemes that avoid **NMPC** and use simple **Linear Model Predictive Control** (**LMPC**) instead.

The dynamics of a humanoid walking on a flat ground can be approximated (in a reasonable way) by a linear model, which is called **Three-dimensional Linear Inverted Pendulum Model** (**3D-LIPM**) [19]. This model is based on two important assumptions. The first one allows to ignore the structure of the robot in order to represent it by an inverted pendulum. The second one restricts motions of pendulum to a plane in order to obtain linear model. Though, this plane can be inclined, henceforth it is assumed to be parallel to the ground surface and intersect the z axis at the height of **CoM** c^z.

The **3D-LIPM** system must be controlled using torques. For convenience, the torques can be expressed through the coordinates of **ZMP** leading to **cart-on-a-table model** [18]. This model allows computation of **ZMP** position using

$$\begin{aligned} z^x &= c^x - h\ddot{c}^x, \\ z^y &= c^y - h\ddot{c}^y, \end{aligned} \tag{2.1}$$

where

$$h = \frac{c^z}{g};$$

(z^x, z^y) and (c^x, c^y) are coordinates of **ZMP** and **CoM** on the horizontal plane; $(\ddot{c}^x, \ddot{c}^y)$ are the accelerations of **CoM** on the horizontal plane; $g = 9.8$ is gravitational acceleration; c^z is the height of the plane, in which motion of the **CoM** is constrained.

Based on the equation (2.1) a discrete-time time-invariant linear dynamical system can be obtained using trivial integration

$$\begin{aligned} \hat{\boldsymbol{c}}_{k+1} &= \boldsymbol{A}\hat{\boldsymbol{c}}_k + \boldsymbol{B}\dddot{\boldsymbol{c}}_k \\ \boldsymbol{z}_k &= \boldsymbol{C}\hat{\boldsymbol{c}}_k. \end{aligned} \tag{2.2}$$

Where the state

$$\hat{\boldsymbol{c}}_k = \begin{bmatrix} c_k^x & \dot{c}_k^x & \ddot{c}_k^x & c_k^y & \dot{c}_k^y & \ddot{c}_k^y \end{bmatrix}^T$$

includes the position, velocity, and acceleration of **CoM**; and the control vector

$$\dddot{\boldsymbol{c}}_k = \begin{bmatrix} \dddot{c}_k^x & \dddot{c}_k^y \end{bmatrix}^T;$$

consists of jerks of **CoM**. State transition and control input matrices are

$$\boldsymbol{A} = \begin{bmatrix} 1 & T & T^2/2 & 0 & 0 & 0 \\ 0 & 1 & T & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & T^2/2 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} T^3/6 & 0 \\ T^2/2 & 0 \\ T & 0 \\ 0 & T^3/6 \\ 0 & T^2/2 \\ 0 & T \end{bmatrix},$$

where T is the length of a discretization step in seconds; and the output matrix is

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & -h & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -h \end{bmatrix}.$$

The system (2.2) is suitable for implementation of an **MPC** controller for **CoM** and **ZMP** trajectory generation.

Another approach to generation of these trajectories is based on analytical solutions of the differential equations (2.1), for example refer to [24, 21]. In this case the **ZMP** trajectory can be represented by a polynomial.

2.4 Nao Overview

The experimental part of this thesis was conducted on a Nao robot. A software module developed for control of this robot is described in Chapter 5.

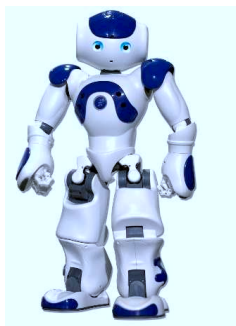


Figure 2.2: The Nao robot. The walking pattern generator that was developed during the work on the thesis was tested on a Nao robot.

Nao (see Figure 2.2) is a small, fully actuated, position-controlled humanoid robot developed by a french company Aldebaran Robotics. We are working with version 3.2 of this robot, which is 58 centimeters tall, weights about 4.5 kilograms and has 25 degrees of freedom. The parameters of the processor installed on this version of Nao is given in Table 2.1.

The operating system of the Nao robots is based on Linux. Various functionalities of Nao are provided by modules, which run within `NaoQI` framework

model name	Geode(TM) Integrated Processor by AMD PCS
cpu MHz	499.903
cache size	128 KB
flags	fpu de pse tsc msr cx8 sep pge cmov clflush mmx mmxext 3dnowext 3dnow

Table 2.1: The Nao CPU information.

on Nao or a computer. Modules communicate with each other transparently through special application programming interface. `NaoQI` framework supports modules written in `C++` or `python`. In order to build modules that must run on the robot, the **Software Development Kit** (**SDK**) is required. The fresh releases of the operating system, the **SDK**, and a comprehensive documentation[1] can be obtained from Aldebaran Robotics. The version of **SDK** and operating system, which were used for development and tests, is 1.12.

The standard software distribution for the Nao robots includes a walking module, which uses the control scheme described in [14].

[1]The Nao software documentation is available on the website of Aldebaran Robotics at `http://www.aldebaran-robotics.com/documentation/index.html`.

Chapter 3
Formulation of Model Predictive Control Problem

Model predictive control [16] is a mature and widely used control paradigm. Its name points to two important characteristics: this strategy is based on the knowledge of the model of the underlying process, and the behavior of the system is predicted over some preview horizon. The goal of **MPC** is to choose a sequence of control inputs for the system over prediction horizon with respect to given objective function and constraints. This goal can be informally interpreted as choosing such control inputs, that do not lead to undesirable effects in the future. The problem is resolved periodically in order to accommodate for the current situation, the preview window is usually shifted in time on each iteration of **MPC**.

Generation of **CoM** and **ZMP** trajectories using an **MPC** scheme was proposed in [18]. This chapter reviews some extensions of the scheme introduced in [34, 9].

3.1 Unconstrained Model Predictive Control Formulation

The original **MPC** problem defined in [18] has finite horizon, a quadratic objective function, and is based on a discrete system (2.2). The objective function penalizes distance from predefined reference points for **ZMP**, the change in state and control. Since this formulation has no inequality constraints an explicit control law (**linear quadratic regulator**) was obtained.

Consider the unconstrained **MPC** based on the system (2.2) where the objective function is defined equivalently to the objective function used in [34].

$$\begin{aligned} \underset{\dddot{\boldsymbol{c}}_0 \ldots \dddot{\boldsymbol{c}}_{N-1}, \hat{\boldsymbol{c}}_1 \ldots \hat{\boldsymbol{c}}_N}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=1}^{N} \|\boldsymbol{C}_p \hat{\boldsymbol{c}}_k - \boldsymbol{z}_k^{ref}\|_{\boldsymbol{Q}}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\dddot{\boldsymbol{c}}_k\|_{\boldsymbol{P}}^2 \\ \text{subject to} \quad & \hat{\boldsymbol{c}}_{k+1} = \boldsymbol{A}\hat{\boldsymbol{c}}_k + \boldsymbol{B}\dddot{\boldsymbol{c}}_k, \end{aligned} \tag{3.1}$$

where

$$\boldsymbol{C}_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

N is the number of sampling intervals in the preview window, matrices $\boldsymbol{Q}$ and $\boldsymbol{P}$ contain gains, $\boldsymbol{z}_k^{ref}$ are reference positions for **ZMP** at discrete sampling time k. The objective function does not penalize the change in the state $\hat{\boldsymbol{c}}_k$, and penalizes the absolute values of control inputs instead of change between subsequent control inputs.

3.2 Introducing the Inequality Constraints

The authors of [34] impose constraints on the position of **ZMP** in order to keep it within the support area and demonstrate that a robot can cope with stronger disturbances in this case. Computation of explicit control law for **MPC** with inequality constraints [2] is not possible here, since the constraints change on each iteration.

Therefore, the application of MPC requires the solution of a quadratic programming problem. The quadratic programming is discussed in Chapter 4. This section is focused on the formulation of **MPC** for **CoM/ZMP** trajectory generation with inequality constraints on **ZMP** positions. This **MPC** is referred to in the thesis as **Model Predictive Control for Walking Motion Generation** (**MPCWMG**).

3.2.1 Double Support Constraints

Note that the support area differs depending on the type of support. **SS** can be easily represented by a rectangle, while **DS** in a general case cannot. Therefore, we need to find a suitable representation of **DS** constraints.

The support area in **DS** is the convex hull of two adjacent **SS**. When the **SS** constraints are rectangular, their convex hull is a polygon as shown in Figure 3.1. In this case the constraints for **SS** and **DS** must be handled differently. We introduce an approximation of this convex hull by a sequence of rectangles, as demonstrated in Figure 3.2. This approach allows to define constraints uniformly, furthermore, it makes possible an extension of the **MPCWMG** to perform footstep repositioning to compensate for disturbances [8].

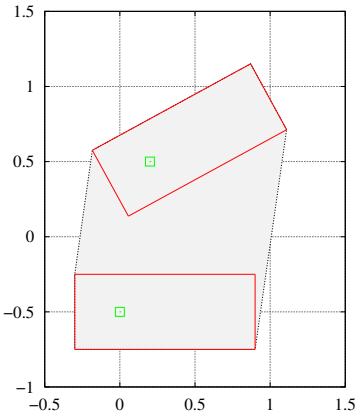


Figure 3.1: A double support represented by a convex hull of two single supports

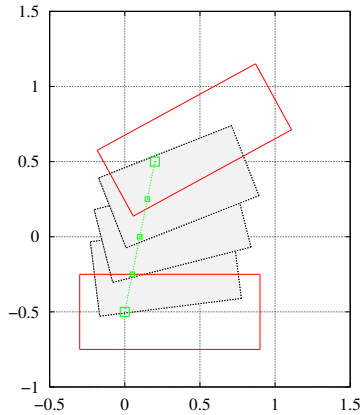


Figure 3.2: A double support approximated by a sequence of rectangles

3.2.2 Definition of the Inequality Constraints

A rectangle can be defined with respect to some reference frame $\mathbb{F}$ by four positive distances $\boldsymbol{d} = \begin{bmatrix} u^x & u^y & -l^x & -l^y \end{bmatrix}^T$ from the origin of $\mathbb{F}$ to the edges of the rectangle. The position and orientation of the rectangle are given by the displacement $\boldsymbol{r}$ and angle of rotation φ of $\mathbb{F}$ with respect to the global reference frame. An example is depicted in Figure 3.3.

All points $\boldsymbol{p}$ lying within the rectangle satisfy

$$\boldsymbol{D}\boldsymbol{R}^T (\boldsymbol{p} - \boldsymbol{r}) \leq \boldsymbol{d},$$

where matrix $\boldsymbol{D}$ and rotation matrix $\boldsymbol{R}$ are defined as

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{I} \\ -\boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}.$$

Thus the constraints for the k-th sampling period in the preview window are

$$\boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{C}\hat{\boldsymbol{c}}_k \leq \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{r}_k, \tag{3.2}$$

where

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & -h & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -h \end{bmatrix}.$$

is the output matrix of the system 2.2 defined in Section 2.3.

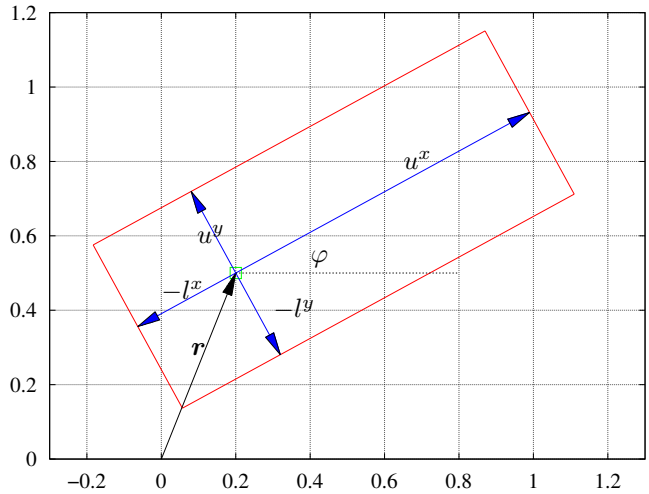


Figure 3.3: Definition of a support rectangle with respect to the global reference frame

3.3 Time-variant Constrained MPC

We can exclude the dynamics of the system (2.2) from the inequality constraints by replacing the position of the **CoM** by the position of the **ZMP** so that

$$\tilde{\boldsymbol{c}} = \begin{bmatrix} z^x & \dot{c}^x & \ddot{c}^x & z^y & \dot{c}^y & \ddot{c}^y \end{bmatrix}^T, \tag{3.3}$$

to obtain

$$\boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{C}_p \tilde{\boldsymbol{c}}_k \leq \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{r}_k. \tag{3.4}$$

The system must be changed accordingly

$$\tilde{\boldsymbol{c}}_{k+1} = \tilde{\boldsymbol{A}}_k \tilde{\boldsymbol{c}}_k + \tilde{\boldsymbol{B}}_k \dddot{\boldsymbol{c}}_k, \tag{3.5}$$

where the state transition and control matrices are defined as

$$\tilde{\boldsymbol{A}}_k = \begin{bmatrix} 1 & T_k & \frac{T_k^2}{2} - \Delta h_k & 0 & 0 & 0 \\ 0 & 1 & T_k & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T_k & \frac{T_k^2}{2} - \Delta h_k \\ 0 & 0 & 0 & 0 & 1 & T_k \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\tilde{\boldsymbol{B}}_k = \begin{bmatrix} \frac{T_k^3}{6} - h_k T_k & 0 \\ \frac{T_k^2}{2} & 0 \\ T_k & 0 \\ 0 & \frac{T_k^3}{6} - h_k T_k \\ 0 & \frac{T_k^2}{2} \\ 0 & T_k \end{bmatrix}.$$

These matrices can be defined as time-invariant, but their parameterization gives us more options for tuning. Variation of the **CoM** height during walk, which can be realized through the appropriate changes of h_k, may have a positive effect on the quality of the gait. Also, the solution of the **Quadratic Program** (**QP**) may be obtained faster, if the sampling time T_k varies in the preview window. Note that the duration of the sampling period in an **MPC** is determined by the control sampling time, and all but the first computed control inputs are usually discarded. Hence, longer sampling periods in the end of the preview window can be used to decrease N, which directly affects the time required for solution, without decreasing the length of this preview window [10].

Now we rewrite the optimization problem (3.1) to reflect the changes in the model

$$\begin{aligned} \underset{\dddot{\tilde{\boldsymbol{c}}}_0 \ldots \ddot{\tilde{\boldsymbol{c}}}_{N-1}; \tilde{\boldsymbol{c}}_1 \ldots \tilde{\boldsymbol{c}}_N}{\text{minimize}} \quad & \sum_{k=1}^{N} \|\tilde{\boldsymbol{c}}_k - \boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|_{\tilde{\boldsymbol{Q}}}^2 + \sum_{k=0}^{N-1} \|\ddot{\tilde{\boldsymbol{c}}}_k\|_{\boldsymbol{P}}^2 \\ \text{subject to} \quad & \tilde{\boldsymbol{c}}_{k+1} = \tilde{\boldsymbol{A}}_k \tilde{\boldsymbol{c}}_k + \tilde{\boldsymbol{B}}_k \ddot{\tilde{\boldsymbol{c}}}_k \\ & \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{C}_p \tilde{\boldsymbol{c}}_k \leq \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{r}_k, \end{aligned} \tag{3.6}$$

where the gain matrices are

$$\tilde{\boldsymbol{Q}} = \begin{bmatrix} \frac{\alpha_g}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\beta_g}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\gamma_g}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\alpha_g}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\beta_g}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\gamma_g}{2} \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} \frac{\eta_g}{2} & 0 \\ 0 & \frac{\eta_g}{2} \end{bmatrix}.$$

The first term in the objective function is

$$\begin{aligned} \sum_{k=1}^{N} \|\tilde{\boldsymbol{c}}_k - \boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|_{\tilde{\boldsymbol{Q}}}^2 &= \sum_{k=1}^{N} \left(\|\tilde{\boldsymbol{c}}_k\|_{\tilde{\boldsymbol{Q}}}^2 + \|\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|_{\tilde{\boldsymbol{Q}}}^2 - 2\tilde{\boldsymbol{c}}_k^T \tilde{\boldsymbol{Q}} \boldsymbol{C}_p^T \boldsymbol{z}_k^{ref} \right) \\ &= \sum_{k=1}^{N} \left(\|\tilde{\boldsymbol{c}}_k\|_{\tilde{\boldsymbol{Q}}}^2 + \|\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|_{\tilde{\boldsymbol{Q}}}^2 - \alpha_g (\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref})^T \tilde{\boldsymbol{c}}_k \right) \\ &= \sum_{k=1}^{N} \left(\|\tilde{\boldsymbol{c}}_k\|_{\tilde{\boldsymbol{Q}}}^2 + \|\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|_{\tilde{\boldsymbol{Q}}}^2 + \boldsymbol{q}_k^T \tilde{\boldsymbol{c}}_k \right), \end{aligned}$$

where $\boldsymbol{q}_k = -\alpha_g(\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref})$.

The terms $\|\boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}\|^2_{\tilde{\boldsymbol{Q}}}$ are constant and can be dropped. Thus we obtain the following formulation **MPCWMG**

$$\begin{aligned} \underset{\dddot{\boldsymbol{c}}_0 \ldots \dddot{\boldsymbol{c}}_{N-1}, \tilde{\boldsymbol{c}}_1 \ldots \tilde{\boldsymbol{c}}_N}{\text{minimize}} \quad & \sum_{k=1}^{N} \|\tilde{\boldsymbol{c}}_k\|^2_{\tilde{\boldsymbol{Q}}} + \sum_{k=0}^{N-1} \|\dddot{\boldsymbol{c}}_k\|^2_{\boldsymbol{P}} + \sum_{k=1}^{N} \boldsymbol{q}_k^T \tilde{\boldsymbol{c}}_k \\ \text{subject to} \quad & \tilde{\boldsymbol{c}}_{k+1} = \tilde{\boldsymbol{A}}_k \tilde{\boldsymbol{c}}_k + \tilde{\boldsymbol{B}}_k \dddot{\boldsymbol{c}}_k \\ & \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{C}_p \tilde{\boldsymbol{c}}_k \leq \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{r}_k. \end{aligned} \tag{3.7}$$

3.4 Generation of an Initial Guess

The **MPC** problem (3.7) is always feasible. This is a useful property for generation of an initial feasible point.

At each iteration of **MPCWMG** the current state of the system (3.5) is assumed to be known. It is also possible to find a set of N points satisfying the inequality constraints (3.4) for each sampling period. This set is a feasible profile for **ZMP**, the next step is to generate control inputs to follow this profile and determine the unknown state variables using a simple iterative procedure.

The k-th control inputs can be computed based on the current state and the next **ZMP** position by multiplying both sides of equation (3.5) by $\boldsymbol{C}_p$

$$\dddot{\boldsymbol{c}}_k = (\boldsymbol{C}_p \tilde{\boldsymbol{B}}_k)^{-1} (\boldsymbol{z}_{k+1} - \boldsymbol{C}_p \tilde{\boldsymbol{A}}_k \tilde{\boldsymbol{c}}_k).$$

Matrix $\boldsymbol{C}_p \tilde{\boldsymbol{B}}_k$ is not invertible when $c_k^z = T_k^2/6$. If the sampling period is less than 0.1 second, then the height of **CoM** of a robot must be lower than approximately 2 millimeters in order to satisfy this equality. Since the sampling period used in simulations and experiments is always below 0.1 second, it is safe to assume that the matrix is always invertible.

Having the k-th control inputs and the current state it is possible to find velocity and acceleration of the next state using equation (3.5).

3.5 Stability of the MPC scheme

When an **MPC** with finite horizon is used, its stability must be considered. It is possible to ensure stability using appropriate modifications of the problem, for a comprehensive review refer to [22]. Stability of the **MPC** formulations presented so far is not guaranteed. Moreover, it is difficult to judge on the stability of the **MPCWMG**, since the problem is altered on each iteration due to the change of the set of the inequality constraints.

Let us neglect the changes of the constraints for the moment and consider the **QP** problem, which is solved on a particular iteration. Note that the discrete systems (2.2) and (3.5) are unstable, since the state transition matrices have repeated eigenvalues equal to 1 and the dimesion of the respective Jordan blocks

is greater than 1. Consequently, it is necessary to impose terminal constraints in order to ensure stability [25]. **QP** with terminal constraints may become infeasible, and generation of an initial guess (Section 3.4) may become more computationally expensive. The stable **MPCWMG** may improve the ability of a robot to cope with external disturbances and may require shorter preview window. However, the effect on the quality of the gait is unknown. Furthermore, the stability of **MPCWMG** does not imply, that the generated **CoM** trajectory can be executed by the controlled robot. Hence, the stability of the **MPCWMG** scheme requires special consideration and experiments, and it was decided to leave it out of the scope of the thesis.

In order to avoid stability issues a "sufficiently long" preview window is used. In [18], where **MPC** problem is not constrained, the explicit control law was computed, and the preview gain was shown to be decreasing to a very small value in about 2 seconds. The built-in walking module for Nao uses preview window of 0.8 second [14]. In a similar way, the necessary preview window length for the developed walking module was found experimentally (it is about 1.6 second, see Chapter 6). Since this time is related to the speed of walk, the number of footsteps is likely to be a better measure of a preview horizon. In our implementation 1.6 second includes three **SS**, or one full oscillation of the **ZMP** trajectory.

Chapter 4
Solving the Quadratic Problem

In Chapter 3 it was pointed out that **MPCWMG** requires solution of a **QP** on each iteration. The theory and algorithms for this class of optimization problems are well developed [27, 4].

We implemented **primal active set** and **primal interior point** methods for the solution of **QP**. Comparison of these two methods with respect to solution of **MPCWMG** is given in Chapter 6. One of these methods can be preferable depending on the **QP**. One iteration of the interior point method is more expensive, but it has polynomial complexity, while the active set method has exponential complexity [29]. Consequently, the active set method can be faster for the problems with relatively small number of inequality constraints [1]. The primal methods are preferred to the dual methods, since each iteration of the former produce a sub-optimal solution that can be used instead of the solution. Note that primal strategies require feasible initial point, which can be easily generated for **MPCWMG** as described in Section 3.4.

The formulations that are used for implementation of active set and logarithmic barrier methods presented in this chapter have certain differences. These differences do not alter the quadratic problem (3.7), but make implementation more convenient or improve performance.

4.1 Fast Solution of MPC Problems

It was demonstrated in [26], that a short sampling period is desirable for control of a walking robot. However, strict time constraints are imposed on the solution of **MPCWMG** in this case. Hence it is necessary to utilize some of the techniques that were developed in order to improve performance of the **MPC** solvers with short sampling periods. In this section only the general techniques are discussed, while more specific optimizations and tuning are described later in this chapter and Chapter 6.

The **MPC** problems are often reformulated in order to reduce the number of decision variables, since it is possible to express the state variables through the

control inputs and eliminate the equality constraints. This process is sometimes referred to as **condensing** [11, 3]. Though the number of decision variables is smaller in this case, the problem (its Hessian and constraints) is less structured. Furthermore, in many cases condensing is performed offline, since it is computationally expensive. This imposes unnecessary limitations, for example, the Hessian is computed for the fixed height of **CoM**. Alternatively, it is possible to solve the problem in the same form as given in equation (3.7) and exploit the structure of the problem. A solver, which is aware of the structure of the problem, can be faster, since the number of floating point operations per one sampling interval is reduced [32, 9, 29]. The approach, which eliminates the equality constraints, in general yields a dense Hessian matrix, while the other one preserves block-diagonal structure of the Hessian. We discriminate these two approaches by calling them **dense** and **sparse**, respectively. Some authors use terms **sequential** and **simultaneous** for this purpose [7]. In [9] we showed how sparse formulation can be applied to **MPCWMG**.

Warm-start techniques can also leverage the performance of the **MPC** solvers. These techniques accelerate the computation of the solution on iteration $k+1$ using the data obtained on the k-th iteration. Warm-start techniques are not adopted in the implementation of the solvers.

In some cases it is impossible to find the solution of **QP** in the available time, but a sub-optimal solution is admissible. Such sub-optimal solution can be obtained, when a primal quadratic programming method (either active set or logarithmic barrier method) is interrupted at some intermediate iteration. This strategy was used in the experiments described in Chapter 6.

4.2 Quadratic Problem in a Matrix Form

In the derivations presented in this chapter it is more convenient to work with the **MPCWMG** (3.7), when **QP** is written in a matrix form.

Let us define the state vector as

$$\boldsymbol{x} = \begin{bmatrix} \tilde{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix},$$

where

$$\tilde{\boldsymbol{v}}_c = \begin{bmatrix} \tilde{\boldsymbol{c}}_1 \\ \vdots \\ \tilde{\boldsymbol{c}}_N \end{bmatrix}, \quad \boldsymbol{v}_u = \begin{bmatrix} \dddot{\boldsymbol{c}}_1 \\ \vdots \\ \dddot{\boldsymbol{c}}_N \end{bmatrix}.$$

The objective function in a matrix form is

$$f\left(\tilde{\boldsymbol{v}}_c, \boldsymbol{v}_u\right) = \begin{bmatrix} \tilde{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix}^T \begin{bmatrix} \tilde{\boldsymbol{H}}_c & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}_u \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix} + \begin{bmatrix} \boldsymbol{g}_c \\ \boldsymbol{0} \end{bmatrix}^T \begin{bmatrix} \tilde{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix},$$

where

$$\tilde{\boldsymbol{H}}_c = \begin{bmatrix} \tilde{\boldsymbol{Q}} & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \tilde{\boldsymbol{Q}} \end{bmatrix}, \quad \boldsymbol{H}_u = \begin{bmatrix} \boldsymbol{P} & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \boldsymbol{P} \end{bmatrix}, \quad \boldsymbol{g} = \begin{bmatrix} \boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_N \end{bmatrix}.$$

The equality constraints are formed based on equation (3.5) and can be expressed as

$$\tilde{\boldsymbol{E}}_c \tilde{\boldsymbol{v}}_c + \tilde{\boldsymbol{E}}_u \boldsymbol{v}_u = \tilde{\boldsymbol{e}}, \tag{4.1}$$

where

$$\tilde{\boldsymbol{E}}_c = \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \tilde{\boldsymbol{A}}_1 & -\boldsymbol{I} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \tilde{\boldsymbol{A}}_{N-1} & -\boldsymbol{I} \end{bmatrix}, \quad \tilde{\boldsymbol{E}}_u = \begin{bmatrix} \tilde{\boldsymbol{B}}_0 & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \tilde{\boldsymbol{B}}_{N-1} \end{bmatrix},$$

and

$$\tilde{\boldsymbol{e}} = \begin{bmatrix} -(\tilde{\boldsymbol{A}}_0 \tilde{\boldsymbol{c}}_0)^T & \boldsymbol{0} & \dots & \boldsymbol{0} \end{bmatrix}^T.$$

Only the vector of states $\tilde{\boldsymbol{v}}_c$ participates in the inequality constraints

$$\begin{bmatrix} \boldsymbol{D}\boldsymbol{R}_1^T \boldsymbol{C}_p & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \boldsymbol{D}\boldsymbol{R}_N^T \boldsymbol{C}_p \end{bmatrix} \tilde{\boldsymbol{v}}_c \leq \begin{bmatrix} \tilde{\boldsymbol{d}}_1 \\ \vdots \\ \tilde{\boldsymbol{d}}_N \end{bmatrix},$$

where

$$\tilde{\boldsymbol{d}}_k = \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T \boldsymbol{r}_k.$$

4.3 Active Set Method

The active set method iteratively tries to guess the inequality constraints that are active at the optimal point. The inequality constraint is called active, if it holds as an equality. The high-level logic of the implemented active set method is shown in Algorithm 1.

On each iteration of this algorithm the **Karush-Kuhn-Tucker conditions** (**KKT**) system is solved to obtain a feasible descent direction $\Delta\boldsymbol{x}$ and a vector of Lagrange multipliers $\boldsymbol{\lambda}$. Then one blocking constraint with the smallest respective step length α is identified. The blocking constraint is added to the active set, and a starting point for the next iteration is computed as $\boldsymbol{x} = \boldsymbol{x} + \alpha\Delta\boldsymbol{x}$. If there are no blocking constraints, the active constraint with the smallest negative Lagrange multiplier is deactivated, and the system is resolved. The algorithm stops, if there are no constraints to add or remove. For a detailed description of this method refer to [27].

Algorithm 1: The active set method

Input : $\boldsymbol{x}$ – initial feasible point
$\boldsymbol{w_i x} \leq \boldsymbol{b}_i$ – inequality constraints $i = 1, \ldots, m$
Output: $\boldsymbol{x}$ – solution of the **QP**

$\boldsymbol{W}^a \leftarrow \emptyset$ /* Indicies of active constraints */
$\boldsymbol{W}^{na} \leftarrow (1, \ldots, m)$ /* Indicies of inactive constraints */
while *true* **do**
  solve the **KKT** system to obtain $\Delta\boldsymbol{x}$ and $\boldsymbol{\lambda}$
  $\alpha \leftarrow 1$ /* The step length $\alpha \in [0; 1]$ */
  $i^a \leftarrow 0$ /* Activated constraint */
  foreach $i \in \boldsymbol{W}^{na}$ **do** /* Find a blocking constraint */
    $\alpha_i \leftarrow$ solution of $\boldsymbol{w}_i(\boldsymbol{x} + \alpha_i \Delta\boldsymbol{x}) = b_i$
    if $\alpha_i < \alpha$ **then**
      $i^a \leftarrow i$
      $\alpha \leftarrow \alpha_i$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\Delta\boldsymbol{x}$
  if $i^a = 0$ **then** /* No constraints to add */
    if $\forall i \in \boldsymbol{W}^a : \boldsymbol{\lambda}_i \geq 0$ **then**
      break /* No constraints to remove */
    $i^d \leftarrow \arg\min_{i \in \boldsymbol{W}^a} \boldsymbol{\lambda}_i$ /* Deactivated constraint */
    $\boldsymbol{W}^a \leftarrow \boldsymbol{W}^a \backslash i^d$
    $\boldsymbol{W}^{na} \leftarrow \boldsymbol{W}^{na} \cup i^d$
  else
    $\boldsymbol{W}^a \leftarrow \boldsymbol{W}^a \cup i^a$

4.3.1 Variable Substitution

A variable substitution that abolishes the linear term in the objective function simplifies the implementation of the solver.

Let us replace the **ZMP** position in the state variable of the system by the distance to the respective reference point to obtain a new state variable

$$\bar{\boldsymbol{c}}_k = \tilde{\boldsymbol{c}}_k - \boldsymbol{C}_p^T \boldsymbol{z}_k^{ref}. \tag{4.2}$$

The system equation (3.5) transforms to

$$\bar{\boldsymbol{c}}_{k+1} + \boldsymbol{C}_p^T \boldsymbol{z}_{k+1}^{ref} = \tilde{\boldsymbol{A}}_k \left(\bar{\boldsymbol{c}}_k + \boldsymbol{C}_p^T \boldsymbol{z}_k^{ref} \right) + \tilde{\boldsymbol{B}}_k \dddot{\boldsymbol{c}}_k. \tag{4.3}$$

The new state vector is $\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{\boldsymbol{v}}_c & \boldsymbol{v}_u \end{bmatrix}^T$, where $\bar{\boldsymbol{v}}_c = \begin{bmatrix} \bar{\boldsymbol{c}}_1^T & \ldots & \bar{\boldsymbol{c}}_N^T \end{bmatrix}^T$.

When the new state vector is plugged into the objective function the linear term disappears

$$f\left(\bar{\boldsymbol{v}}_c, \boldsymbol{v}_u\right) = \begin{bmatrix} \bar{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix}^T \begin{bmatrix} \tilde{\boldsymbol{H}}_c & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}_u \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix}.$$

The matrices participating in the equality constraints (4.1) are not altered. Only the vector $\tilde{\boldsymbol{e}}$ must be replaced by

$$\bar{\boldsymbol{e}} = \begin{bmatrix} -\tilde{\boldsymbol{A}}_0 \left(\bar{\boldsymbol{c}}_0 + \boldsymbol{C}_p^T \boldsymbol{z}_0^{ref}\right) + \boldsymbol{C}_p^T \boldsymbol{z}_1^{ref} \\ -\tilde{\boldsymbol{A}}_1 \boldsymbol{C}_p^T \boldsymbol{z}_1^{ref} + \boldsymbol{C}_p^T \boldsymbol{z}_2^{ref} \\ \cdots \\ -\tilde{\boldsymbol{A}}_{N-1} \boldsymbol{C}_p^T \boldsymbol{z}_{N-1}^{ref} + \boldsymbol{C}_p^T \boldsymbol{z}_N^{ref} \end{bmatrix}.$$

The inequality constraints are changed to

$$\boldsymbol{D}\boldsymbol{R}_k^T \bar{\boldsymbol{c}}_k \leq \boldsymbol{d}_k + \boldsymbol{D}\boldsymbol{R}_k^T (\boldsymbol{r}_k - \boldsymbol{z}_k^{ref}).$$

The inequality constraints prevent coordinates of **ZMP** from leaving the rectangular support polygons. We can interpret them as a point $\boldsymbol{R}_k^T \boldsymbol{C}_p \bar{\boldsymbol{c}}_k$ having lower and upper bounds. Since lower and upper bounds cannot be activated simultaneously, checking for activated constraints can be simplified.

4.3.2 Solution of the KKT system

The **KKT** system

$$\begin{bmatrix} 2\boldsymbol{H} & \boldsymbol{E}^T \\ \boldsymbol{E} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{init} + \Delta\boldsymbol{x} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{e}} \end{bmatrix} \tag{4.4}$$

is solved using block elimination complement, which is convenient for solving a sparse **QP** [12]. Since the Hessian matrix is diagonal, it can be easily inverted assuming that all gains are strictly greater than 0.

Then the Lagrange multipliers and the descent direction can be found from

$$\frac{1}{2}\boldsymbol{E}\boldsymbol{H}^{-1}\boldsymbol{E}^T\boldsymbol{\nu} = \boldsymbol{S}\boldsymbol{\nu} = -\boldsymbol{E}\boldsymbol{x}_{init} = \boldsymbol{s},$$

$$\Delta\boldsymbol{x} = -\boldsymbol{x}_{init} - \frac{1}{2}\boldsymbol{H}^{-1}\boldsymbol{E}^T\boldsymbol{\nu},$$

where $\boldsymbol{S}$ is the negated Schur complement, which is also referred to simply as Schur complement in the thesis.

Schur Complement

The negated Schur complement for the **KKT** system (4.4) is

$$\begin{aligned}\boldsymbol{S} &= \frac{1}{2}\boldsymbol{E}\boldsymbol{H}^{-1}\boldsymbol{E}^T = \frac{1}{2}\left[\tilde{\boldsymbol{E}}_c\tilde{\boldsymbol{E}}_u\right]\begin{bmatrix}\tilde{\boldsymbol{H}}_c & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{H}_u\end{bmatrix}^{-1}\begin{bmatrix}\tilde{\boldsymbol{E}}_c^T\\ \tilde{\boldsymbol{E}}_u^T\end{bmatrix}\\ &= \frac{1}{2}\tilde{\boldsymbol{E}}_c\tilde{\boldsymbol{H}}_c^{-1}\tilde{\boldsymbol{E}}_c^T + \frac{1}{2}\tilde{\boldsymbol{E}}_u\boldsymbol{H}_u^{-1}\tilde{\boldsymbol{E}}_u^T.\end{aligned}$$

In Appendix A it is demonstrated that $\boldsymbol{S}$ has a block-diagonal structure with the blocks defined as

$$\begin{aligned}2\boldsymbol{S}_{11} &= \tilde{\boldsymbol{Q}}^{-1} + \tilde{\boldsymbol{B}}_0\boldsymbol{P}^{-1}\tilde{\boldsymbol{B}}_0^T,\\ 2\boldsymbol{S}_{kk} &= \boldsymbol{A}_{k-1}\tilde{\boldsymbol{Q}}^{-1}\boldsymbol{A}_{k-1}^T + \tilde{\boldsymbol{Q}}^{-1} + \tilde{\boldsymbol{B}}_{k-1}\boldsymbol{P}^{-1}\tilde{\boldsymbol{B}}_{k-1}^T,\\ 2\boldsymbol{S}_{k,k+1} &= \boldsymbol{S}_{k+1,k}^T = -\tilde{\boldsymbol{Q}}^{-1}\boldsymbol{A}_k^T.\end{aligned} \tag{4.5}$$

Cholesky Factorization of the Schur Complement

In order to determine the Lagrange multipliers from equation (4.4) we have to solve a linear system. The matrix $\boldsymbol{S}$ is positive definite, since the inverted Hessian matrix is positive definite by construction and the rows of the matrix of constraints are linearly independent [13]. The rows of the matrix of constraints remain linearly independent after addition of active constraints due to the structure of equality and inequality constraints. Hence, it is possible to solve the linear system using Cholesky factorization $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$, where $\boldsymbol{L}$ is lower triangular

$$\boldsymbol{L} = \begin{bmatrix}\boldsymbol{L}_{11} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{L}_{32} & \boldsymbol{L}_{33} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{L}_{N-1,N-1} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{L}_{N,N-1} & \boldsymbol{L}_{NN}\end{bmatrix}.$$

Directly from observation of equations (4.5) and the structure of $\boldsymbol{L}$

$$\begin{aligned}\boldsymbol{S}_{11} &= \boldsymbol{L}_{11}\boldsymbol{L}_{11}^T,\\ \boldsymbol{S}_{12} &= \boldsymbol{S}_{21}^T = \boldsymbol{L}_{11}\boldsymbol{L}_{21}^T\\ \boldsymbol{S}_{22} &= \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T + \boldsymbol{L}_{22}\boldsymbol{L}_{22}^T,\end{aligned}$$

and so on. $\boldsymbol{L}_{21}^T$ is computed by forward substitution, forming $\boldsymbol{L}_{22}$ requires the computation of the Cholesky factors of $\boldsymbol{S}_{22} - \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T$.

Note that the state and control matrices in the system (as introduced in Section 3.3) are decoupled and this property is preserved in the Schur complement and its Cholesky factor.

Computation of the Step Direction

Let $\boldsymbol{\nu}_e$ and $\boldsymbol{\lambda}$ be the Lagrange multipliers associated with the equality and inequality constraints, respectively, $\boldsymbol{A}_W$ be the matrix of normals of active constraints, then

$$\Delta \boldsymbol{x} = -\boldsymbol{x}_{init} - \frac{1}{2}\boldsymbol{H}^{-1}\left(\boldsymbol{E}^T\boldsymbol{\nu_e} + \boldsymbol{A}_W^T\boldsymbol{\lambda}\right).$$

Multiplication by the inverted Hessian is trivial since it is diagonal.

4.3.3 Resolving the Problem after Changes of the Active Set

Update of the Cholesky Factor after Addition of a Constraint

The updated Schur complement, which is denoted as $\boldsymbol{S}^+$, is

$$\boldsymbol{S}^+ = \frac{1}{2}\begin{bmatrix}\boldsymbol{C}\\ \boldsymbol{a}_i^T\end{bmatrix}\boldsymbol{H}^{-1}\begin{bmatrix}\boldsymbol{C}^T & \boldsymbol{a}_i\end{bmatrix} = \frac{1}{2}\begin{bmatrix}\boldsymbol{C}\boldsymbol{H}^{-1}\boldsymbol{C}^T & \boldsymbol{C}\boldsymbol{H}^{-1}\boldsymbol{a}_i\\ \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{C}^T & \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{a}_i\end{bmatrix},$$

where

$$\boldsymbol{C} = \begin{bmatrix}\boldsymbol{E}\\ \boldsymbol{A}_W\end{bmatrix},$$

and the new new row of $\boldsymbol{S}^+$ is

$$\boldsymbol{s_a}^T = \frac{1}{2}\begin{bmatrix}\boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{E}^T & \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{A}_W^T & \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{a}_i\end{bmatrix}.$$

Where

$$\frac{1}{2}\boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{a}_i = \frac{1}{\alpha_g},$$

$\boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{E}^T$ has up to 4 nonzero elements, $\boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{A}_W^T$ is a vector of zeros, since the activated inequality constraints are always orthogonal (the constrained regions are rectangles). The total number of non-zero elements in the new row of the Schur complement is 5 or 3 (for the last state in the preview window).

Consider the updated Cholesky factor

$$\boldsymbol{L}^+ = \begin{bmatrix}\boldsymbol{L}\\ \boldsymbol{l}^T\end{bmatrix},$$

where $\boldsymbol{l}^T$ is the row to be added. Note that

$$\boldsymbol{S}^+ = \boldsymbol{L}^+(\boldsymbol{L}^+)^T = \begin{bmatrix}\boldsymbol{L}\boldsymbol{L}^T & \boldsymbol{L}\boldsymbol{l}\\ \boldsymbol{l}^T\boldsymbol{L}^T & \boldsymbol{l}^T\boldsymbol{l}\end{bmatrix} = \begin{bmatrix}\boldsymbol{C}\boldsymbol{H}^{-1}\boldsymbol{C}^T & \boldsymbol{C}\boldsymbol{H}^{-1}\boldsymbol{a}_i\\ \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{C}^T & \boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{a}_i\end{bmatrix}.$$

Consequently,

$$\begin{bmatrix}\boldsymbol{L}\\ \boldsymbol{l}^T\end{bmatrix}\boldsymbol{l} = \begin{bmatrix}\boldsymbol{L} & \boldsymbol{0}\\ \boldsymbol{l}_L^T & \ell\end{bmatrix}\begin{bmatrix}\boldsymbol{l}_L\\ \ell\end{bmatrix} = \boldsymbol{s}_a, \quad \ell = \sqrt{\boldsymbol{a}_i^T\boldsymbol{H}^{-1}\boldsymbol{a}_i - \boldsymbol{l}_L^T\boldsymbol{l}_L},$$

thus the computation of the new row of $\boldsymbol{L}^+$ amounts to a forward substitution and a dot product of two vectors. Furthermore, due to the structure of the inequality constraints, the first $N(i-1)$ elements of $\boldsymbol{s}_a$ corresponding to constraint for the i-th sampling period are zeros and full forward substitution is not necessary.

Update of the Cholesky Factor after Removal of a Constraint

Imagine that the i-th inequality constraint was selected for removal, then the corresponding line and column must be removed from the matrix $\boldsymbol{S}$. This can be achieved by moving these lines to the end and to the right of $\boldsymbol{S}$ using a row-interchanging permutation matrix $\boldsymbol{U}$:

$$\boldsymbol{U}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{L}\boldsymbol{L}^T\boldsymbol{U}^T = (\boldsymbol{U}\boldsymbol{L})(\boldsymbol{U}\boldsymbol{L})^T.$$

The matrix $\boldsymbol{U}\boldsymbol{L}$ is not lower triangular

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{b}^T & p & \boldsymbol{0} \\ \boldsymbol{L}_2 & \boldsymbol{d} & \boldsymbol{L}_3 \end{bmatrix}, \quad \boldsymbol{U}\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{L}_2 & \boldsymbol{d} & \boldsymbol{L}_3 \\ \boldsymbol{b}^T & p & \boldsymbol{0} \end{bmatrix},$$

where $\begin{bmatrix} \boldsymbol{b}^T & p & \boldsymbol{0} \end{bmatrix}$ is a row, that must be removed. The square matrix $\boldsymbol{L}_1$ of $(6N+i-1)\times(6N+i-1)$ size and matrix $\boldsymbol{L}_2$ of $(m_a-i)\times(6N+i-1)$ size do not depend on the removed row. The matrix $\begin{bmatrix} \boldsymbol{d} & \boldsymbol{L}_3 \end{bmatrix}$ of $(m_a-i)\times(m_a-i+1)$ size can be transformed to the triangular form using a sequence of m_a-1 Givens rotation matrices (refer to [13] for background). Each rotation matrix in this sequence alters two columns of $\boldsymbol{L}_3$ starting from the left. The rotation matrices eliminate each other, which can be demonstrated with one rotation matrix $\boldsymbol{G}$

$$\begin{aligned} \left(\boldsymbol{U}\boldsymbol{L}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} \end{bmatrix}\right)\left(\boldsymbol{U}\boldsymbol{L}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} \end{bmatrix}\right)^T &= \boldsymbol{U}\boldsymbol{L}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} \end{bmatrix}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G} \end{bmatrix}^T \boldsymbol{L}^T\boldsymbol{U}^T \\ &= \boldsymbol{U}\boldsymbol{L}\boldsymbol{L}^T\boldsymbol{U}^T. \end{aligned}$$

The Cholesky decomposition is unique: given a positive-definite matrix $\boldsymbol{S}$, there is only one lower triangular matrix $\boldsymbol{L}$ with strictly positive diagonal entries such that $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$. The procedure described above may produce negative diagonal entries. The sign of the diagonal element can be changed using multiplication by an identity matrix having -1 on the respective position. The signs of all elements of the column, where negative element is located, are changed.

Computation of the Lagrange Multipliers after Addition of a Constraint

Consider the addition of the i-th inequality constraint with the normal $\boldsymbol{a}_i^T$ to the active set. Let vector $\boldsymbol{s} = -\boldsymbol{C}\boldsymbol{x}$. Then, after the addition of the new constraint

$$\boldsymbol{s}^+ = -\begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{a}_i^T \end{bmatrix}(\boldsymbol{x}+\alpha\Delta\boldsymbol{x}) = -\begin{bmatrix} \boldsymbol{C}\boldsymbol{x} \\ \boldsymbol{a}_i^T(\boldsymbol{x}+\alpha\Delta\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{s} \\ s_n \end{bmatrix}.$$

Note that $\alpha \boldsymbol{C}\Delta\boldsymbol{x} = \boldsymbol{0}$, because $\Delta\boldsymbol{x}$ is in the null space of the normals of the constraints stored in $\boldsymbol{C}$.

After $\boldsymbol{L}$ was changed due to addition of inequality constraint, the solution of the linear system does not require full forward substitution. Let $\boldsymbol{z} = \boldsymbol{L}^T\boldsymbol{\nu}$ then $\boldsymbol{L}\boldsymbol{z} = \boldsymbol{s}$, then after the constraint is added

$$\underbrace{\begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{l}_p^T & \ell \end{bmatrix}}_{\boldsymbol{L}^+} \underbrace{\begin{bmatrix} \boldsymbol{z} \\ z_n \end{bmatrix}}_{\boldsymbol{z}^+} = \underbrace{\begin{bmatrix} \boldsymbol{s} \\ s_n \end{bmatrix}}_{\boldsymbol{s}^+},$$

where $\boldsymbol{l}^T = \begin{bmatrix} \boldsymbol{l}_p^T & \ell \end{bmatrix}$ is a row appended to $\boldsymbol{L}$ to obtain $\boldsymbol{L}^+$. Since

$$\boldsymbol{l}_p^T\boldsymbol{z} + bz_n = s_n,$$

we can compute (note that $\ell \neq 0$)

$$z_n = \frac{s_n - \boldsymbol{l}^T\boldsymbol{z}}{\ell}.$$

Hence, forming $\boldsymbol{z}^+$ amounts to performing one dot product.

Computation of the Lagrange Multipliers after Removal of a Constraint

Consider the system before removal of a constraint

$$\boldsymbol{L}\boldsymbol{z} = \begin{bmatrix} \boldsymbol{L}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{b}^T & p & \boldsymbol{0} \\ \boldsymbol{L}_2 & \boldsymbol{d} & \boldsymbol{L}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_c \\ z_p \\ \boldsymbol{z}_o \end{bmatrix} = \boldsymbol{s} = \begin{bmatrix} \boldsymbol{s}_1 \\ s_p \\ \boldsymbol{s}_2 \end{bmatrix},$$

where $\boldsymbol{z}_c, \boldsymbol{s}_1, \boldsymbol{s}_2$ remain constant; z_p and s_p must be removed; $\boldsymbol{z}_o$ must be updated. From

$$\boldsymbol{L}_2\boldsymbol{z}_c + \boldsymbol{d}z_p + \boldsymbol{L}_3\boldsymbol{z}_o = \boldsymbol{s}_2$$

we can obtain vector $\boldsymbol{s}_2 - \boldsymbol{L}_2\boldsymbol{z}_c$, which is not affected by the removal of a constraint.

The updated system is

$$\begin{bmatrix} \boldsymbol{L}_1 & \boldsymbol{0} \\ \boldsymbol{L}_2 & \boldsymbol{L}_{3,u} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_c \\ \boldsymbol{z}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{bmatrix}.$$

The updated part of vector $\boldsymbol{z}$ can be found using forward substitution from

$$\boldsymbol{L}_{3,u}\boldsymbol{z}_u = \boldsymbol{s}_2 - \boldsymbol{L}_2\boldsymbol{z}_c.$$

4.4 Logarithmic Barrier Method

The logarithmic barrier method is a relatively simple interior-point algorithm. On each iteration of this method a logarithmic barrier is added to the objective function, then a feasible descent direction is obtained from the solution of the **KKT** system, which is formed using Taylor approximation of the new objective function. The implementation of solver, which was developed during the work on this thesis, follows Algorithm 2. A comprehensive description of the logarithmic barrier method is given in [4].

Algorithm 2: The logarithmic barrier method

Input : $\boldsymbol{x}$ – initial feasible point
$t > 0$ – multiplier of logarithmic barrier term
$\mu > 1$ – increase rate of t
ϵ – tolerance
m – number of inequality constraints
Output: $\boldsymbol{x}$ – solution of the **QP**

```
while true do                                          /* External loop */
    while true do                                      /* Internal loop */
        form KKT system using Taylor approximation of φ
        Δx ← solution of KKT system
        if Δx^T ∇²φ Δx < ε then
            break                          /* The decrement is too small */
        α ← step length found using backtracking search
        if α < ε then
            break                      /* The step length is too small */
        x ← x + αΔx
    t ← tμ
    if m/t < ε then
        break                          /* The duality gap is too small */
```

4.4.1 Review of the Logarithmic Barrier Method

Consider the following **QP**

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{g}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{E}\boldsymbol{x} = \boldsymbol{e} \\ & \boldsymbol{W}\boldsymbol{x} - \boldsymbol{d} \leq \boldsymbol{0}. \end{aligned}$$

After addition of the logarithmic barrier term the objective function changes to

$$\phi(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^T \boldsymbol{x} - \frac{1}{t}\sum_{i=1}^{m} \ln\left(-\boldsymbol{W}_i \boldsymbol{x} + \boldsymbol{d}_i\right).$$

The Taylor approximation of ϕ near some feasible point $\boldsymbol{x}$ is

$$\phi(\boldsymbol{x} + \Delta\boldsymbol{x}) = \phi(\boldsymbol{x}) + \nabla\phi(\boldsymbol{x})\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}\nabla^2\phi(\boldsymbol{x})\Delta\boldsymbol{x},$$

where

$$\nabla\phi(\boldsymbol{x}) = 2\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} + \frac{1}{t}\sum_{i=1}^{m}\left(\frac{1}{-\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{d}_i}\boldsymbol{W}_i^T\right),$$

$$\nabla^2\phi(\boldsymbol{x}) = 2\boldsymbol{H} + \frac{1}{t}\sum_{i=1}^{m}\left(\frac{1}{(-\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{d}_i)^2}\boldsymbol{W}_i^T\boldsymbol{W}_i\right).$$

Then **KKT** system can be derived from the Taylor approximation of ϕ

$$\begin{bmatrix} \nabla^2\phi(\boldsymbol{x}) & \boldsymbol{E}^T \\ \boldsymbol{E} & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} \Delta\boldsymbol{x} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} -\nabla\phi(\boldsymbol{x}) \\ \boldsymbol{0} \end{bmatrix},$$

which can be solved using block substitution complement as

$$\Delta\boldsymbol{x} = (\nabla^2\phi(\boldsymbol{x}))^{-1}(-\nabla\phi(\boldsymbol{x}) - \boldsymbol{E}^T\boldsymbol{\omega}),$$
$$\boldsymbol{E}(\nabla^2\phi(\boldsymbol{x}))^{-1}\boldsymbol{E}^T\boldsymbol{\omega} = -\boldsymbol{E}(\nabla^2\phi(\boldsymbol{x}))^{-1}\nabla\phi(\boldsymbol{x}).$$

Again, $\boldsymbol{\omega}$ can be found using Cholesky decomposition. Since ϕ is approximated near $\boldsymbol{x}$, the system must be iteratively resolved until $\Delta\boldsymbol{x}$ is small enough.

4.4.2 Variable Substitution

It is more convenient to implement interior point method for a **QP**, which has only simple bounds. It is possible to obtain **MPCWMG** with simple bounds by performing a variable substitution described in this section.

Each state is constrained by four inequalities

$$\boldsymbol{D}\boldsymbol{R}_k^T\boldsymbol{C}_p\tilde{\boldsymbol{c}}_k \leq \tilde{\boldsymbol{d}}_k.$$

Let the new state variable be $\boldsymbol{p}_k = \bar{\boldsymbol{R}}_k^T\tilde{\boldsymbol{c}}_k$, where

$$\bar{\boldsymbol{R}}_k = \begin{bmatrix} \cos\varphi_k & 0 & 0 & -\sin\varphi_k & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \sin\varphi_k & 0 & 0 & \cos\varphi_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The four elements of $\tilde{\boldsymbol{d}}_k$ are

$$\tilde{\boldsymbol{d}}_k = \begin{bmatrix} u_k^x & u_k^y & -l_k^x & -l_k^y \end{bmatrix}^T,$$

where u_k^x, u_k^y and l_k^x, l_k^y are upper and lower bounds, respectively.

Then the inequality constraints for state $\boldsymbol{p}_k$ are changed to

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \boldsymbol{p}_k \leq \begin{bmatrix} u_k^x \\ u_k^y \\ -l_k^x \\ -l_k^y \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} p_k^x \\ p_k^y \\ -p_k^x \\ -p_k^y \end{bmatrix} \leq \begin{bmatrix} u_k^x \\ u_k^y \\ -l_k^x \\ -l_k^y \end{bmatrix}.$$

The system equation must be changed accordingly

$$\bar{\boldsymbol{R}}_{k+1}\boldsymbol{p}_{k+1} = \tilde{\boldsymbol{A}}_k\bar{\boldsymbol{R}}_k\boldsymbol{p}_k + \tilde{\boldsymbol{B}}_k\ddot{\boldsymbol{c}}_k.$$

Consequently, the equality constraints are also changed

$$\bar{\boldsymbol{E}}_c\bar{\boldsymbol{v}}_c + \tilde{\boldsymbol{E}}_u\boldsymbol{v}_u = \bar{\boldsymbol{e}},$$

where

$$\bar{\boldsymbol{e}} = \begin{bmatrix} -\boldsymbol{A}_0\bar{\boldsymbol{R}}_0\boldsymbol{p}_0 \\ \boldsymbol{0} \end{bmatrix},$$

and

$$\bar{\boldsymbol{E}}_c = \begin{bmatrix} -\bar{\boldsymbol{R}}_1 & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_1\bar{\boldsymbol{R}}_1 & -\bar{\boldsymbol{R}}_2 & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_2\bar{\boldsymbol{R}}_2 & -\bar{\boldsymbol{R}}_3 & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{A}_{N-1}\bar{\boldsymbol{R}}_{N-1} & -\bar{\boldsymbol{R}}_N \end{bmatrix}.$$

4.4.3 Objective Function of a QP with Simple Bounds

Consider the objective function of **MPCWMG** with simple bounds after addition of the logarithmic barrier

$$\phi(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^T\boldsymbol{x} - \frac{1}{t}\sum_{i=1}^{N}(\ln(-p_k^x + u_k^x) + \ln(-p_k^y + u_k^y) + \ln(p_k^x - l_k^x) + \ln(p_k^y - l_k^y)).$$

The derivative $\nabla\phi(\boldsymbol{x})$ can be computed as

$$\nabla\phi(\boldsymbol{x}) = 2\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} + \frac{1}{t}\boldsymbol{o},$$

where

$$\boldsymbol{o} = \begin{bmatrix} \boldsymbol{o}_1^T & \dots & \boldsymbol{o}_N^T & \boldsymbol{0} \end{bmatrix}^T$$

and

$$\boldsymbol{o}_k = \begin{bmatrix} \frac{1}{-p_k^x + u_k^x} - \frac{1}{p_k^x - l_k^x} \\ 0 \\ 0 \\ \frac{1}{-p_k^y + u_k^y} - \frac{1}{p_k^y - l_k^y} \\ 0 \\ 0 \end{bmatrix}.$$

The second derivative of $\phi(\boldsymbol{x})$ is

$$\nabla^2 \phi(\boldsymbol{x}) = 2\boldsymbol{H} + \frac{1}{t}\boldsymbol{O},$$

where

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{O}_1 & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \dots & \boldsymbol{O}_N & \\ \boldsymbol{0} & \dots & & \boldsymbol{0} \end{bmatrix}$$

and

$$\boldsymbol{O}_k = \begin{bmatrix} \frac{1}{(-p_k^x + u_k^x)^2} + \frac{1}{(p_k^x - l_k^x)^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{(-p_k^y + u_k^y)^2} + \frac{1}{(p_k^y - l_k^y)^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

4.4.4 Schur Complement

The Cholesky factorization of the negated Schur complement $\bar{\boldsymbol{E}}(\nabla^2 \phi(\boldsymbol{x}))^{-1} \bar{\boldsymbol{E}}^T$ must be performed on each iteration of the logarithmic barrier method, since the function $\phi(\boldsymbol{x})$ is approximated near the starting point of each iteration.

The negated Schur complement is defined as

$$\begin{aligned} \bar{\boldsymbol{E}}(\nabla^2 \phi(\boldsymbol{x}))^{-1} \bar{\boldsymbol{E}}^T &= \bar{\boldsymbol{E}}_c(\nabla^2 \phi(\boldsymbol{x}))^{-1} \bar{\boldsymbol{E}}_c^T + \tilde{\boldsymbol{E}}_u(\nabla^2 \phi(\boldsymbol{x}))^{-1} \tilde{\boldsymbol{E}}_u^T \\ &= \bar{\boldsymbol{E}}_c(\nabla^2 \phi(\boldsymbol{x}))^{-1} \bar{\boldsymbol{E}}^T + \tilde{\boldsymbol{E}}_u \boldsymbol{H}_u^{-1} \tilde{\boldsymbol{E}}_u^T. \end{aligned}$$

$$\bar{\boldsymbol{E}}_c(\nabla^2 \phi(\boldsymbol{x}))^{-1} \bar{\boldsymbol{E}}^T = \begin{bmatrix} \boldsymbol{M}_{11} & -\boldsymbol{M}_{11}\boldsymbol{A}^T & \boldsymbol{0} \\ -\boldsymbol{A}\boldsymbol{M}_{11} & \boldsymbol{A}\boldsymbol{M}_{11}\boldsymbol{A}^T + \boldsymbol{M}_{22} & -\boldsymbol{M}_{22}\boldsymbol{A}^T \\ \boldsymbol{0} & -\boldsymbol{A}\boldsymbol{M}_{22} & \boldsymbol{A}\boldsymbol{M}_{22}\boldsymbol{A}^T + \boldsymbol{M}_{33} \end{bmatrix},$$

where $\boldsymbol{M}_{kk} = \bar{\boldsymbol{R}}_k \bar{\boldsymbol{Q}}_k^{-1} \bar{\boldsymbol{R}}_k^T$ and $\bar{\boldsymbol{Q}}_k = \tilde{\boldsymbol{Q}} + \boldsymbol{O}^k$.

The Cholesky factorization is performed in the same way as described in Section 4.3.2. Note that due to addition of $\boldsymbol{O}^k$, the Cholesky factor is not so well structured as in the case of the active set method.

Chapter 5
Implementation of a Walking Module for Nao

During the work on the thesis two solvers for **MPCWMG**, a simple footstep pattern generator, an inverse kinematics library, and a walking module for Nao robots were developed[1]. Instructions on compilation and installation are distributed with the sources.

The architecture of the walking module is schematically depicted in Figure 5.1.

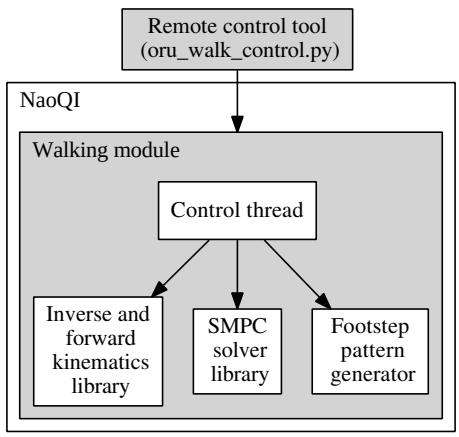


Figure 5.1: Architecture of the walking module

[1] All sources are publicly available at `https://github.com/asherikov/`.

The walking module runs within the `NaoQI` framework on a Nao robot. Commands are sent to the module using a simple command line control tool from a personal computer.

When walk is started the module reads parameters from a configuration file and spawns a control thread, which is periodically activated in order to determine the joint commands. The necessary joint angles are determined based on the desired position of the **CoM** and poses of the feet provided by **MPCWMG** solvers and footstep pattern generator, respectively. The functions of each submodule are described in more detail in this chapter.

Note that some implementations [14] assume that the **CoM** is fixed to the torso, hence the trajectory of the **CoM** is tracked by some point on the torso. We do not make such assumption and control the real **CoM**.

5.1 Sparse MPC Solver

The purpose of the **MPCWMG** is to generate a trajectory for the **CoM**. To be precise, a trajectory for the **ZMP** is generated first, then the **CoM** positions are computed from it. The trajectories are shown in Figure 5.2. In order to generate trajectories the reference **ZMP** points must be given, in **SS** they are placed in the middle of the constraining rectangle, in a **DS** the reference **ZMP** point of the closest **SS** is used.

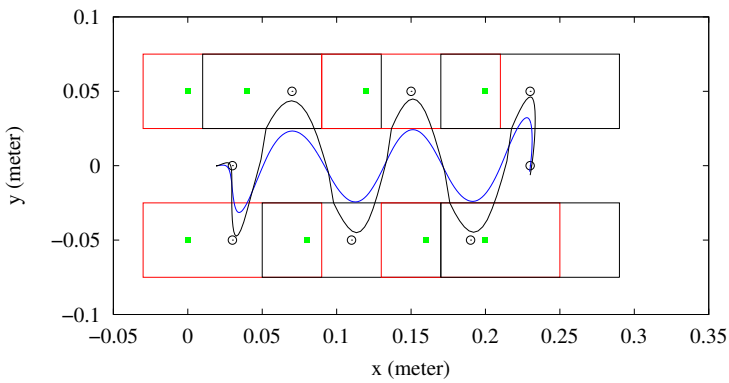


Figure 5.2: Red and black rectangles represent **SS**, **DS** are omitted. Blue and black curves show trajectories of **CoM** and **ZMP**, respectively. Black circles stand for reference **ZMP** positions. Green boxes are the footstep reference points. The data was obtained during simulation.

Both active set (Section 4.3) and logarithmic barrier (Section 4.4) methods were developed for solving the **QP**. One of them can be selected by appropriately setting options in the configuration file. In particular, these options can be used to control the mechanisms that limit the execution time of the solver. The execution time of active set method can be controlled by imposing a limit

on the number of activated constraints and disabling the deactivation of constraints. The number of iterations of the logarithmic barrier method can be also explicitly limited.

5.2 Footstep Pattern Generator

The functionality of the footstep pattern generator is two-fold. It provides functions to predefine footsteps and generate swing foot trajectories during walk.

The positions and orientations of the footsteps must be defined before walk is started, an example of footstep sequence is depicted in Figure 5.2.

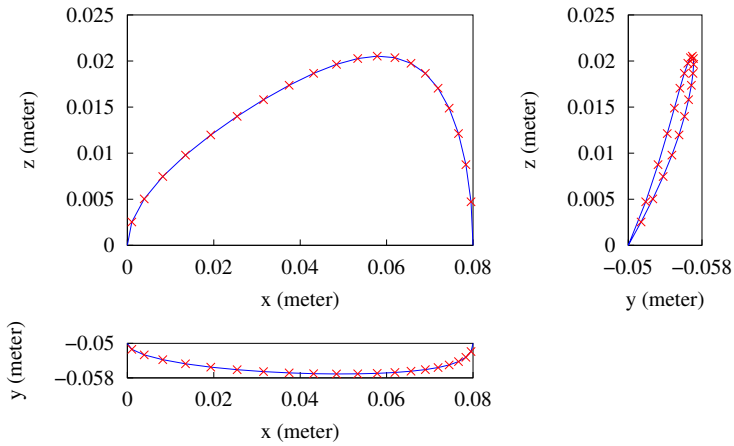


Figure 5.3: Right swing foot trajectory in three projections. Red crosses show points that correspond to sampling intervals of 20 milliseconds. The data was obtained during simulation.

The foot trajectories are generated using Bezier curves. Parameters for the curves were found experimentally. The three projections of a right foot trajectory produced during straight walk are shown in Figure 5.3, the trajectory of the left foot is symmetric.

5.3 Inverse and Forward Kinematics Library

Inverse and forward kinematics library provide functions that solve **Forward Kinematics** (**FK**) and **Inverse Kinematics** (**IK**). In the sources it is also referred to as **Inverse Geometrical Model** (**IGM**), where the word "geometrical" is used instead of "kinematics" to emphasize, that the library works with positions of actuators and not with their motions.

5.3.1 Algorithms

From a mechanical point of view a humanoid robot is a chain of $n+1$ rigid bodies (**links**) connected by n **joints**. One of the extremities of this chain is often referred to as the base and some of the others as end-effectors. Motions of a robot are compositions of elementary motions of links with respect to each other. Motion can be represented in two spaces: **joint space** and **operational space**, in the former one the coordinates are n joint angles, in the latter – positions and orientations of end-effectors with respect to the base. Joint and operational spaces are related by **FK** and **IK**.

Let $\boldsymbol{q} \in \mathbb{R}^n$ be the joint space coordinates, and $\boldsymbol{r} \in \mathbb{R}^m$ be the operational space coordinates.

The purpose of the **FK** is determination of position and orientation of a frame attached to an end-effector given the reference frame attached to the base of the chain and a set of joint angles. It is performed by solving

$$\boldsymbol{r} = f_r(\boldsymbol{q}).$$

The **IK** problem is opposite to the **FK** problem. **IK** finds the set of joint angles that realizes the desired pose of an end-effector using inverse function of f_r

$$\boldsymbol{q} = f_r^{-1}(\boldsymbol{r}).$$

There are two approaches to solve the **IK** problem: analytical (closed-form) and numerical. The **IK** is a typical nonlinear problem and it is not always possible to find a closed-form solution. In such cases numerical methods can be used instead. A numerical solution of **IK** can be found in a several ways using different iterative schemes. The implementation of **IK** for Nao used in our work is similar to the method described in [35].

The Taylor approximation of the **FK** function can be expressed as

$$\boldsymbol{r} = f_r(\boldsymbol{q}) \approx f_r(\boldsymbol{q}_k) + \boldsymbol{J}_r(\boldsymbol{q}_k)(\boldsymbol{q} - \boldsymbol{q}_k), \tag{5.1}$$

where $\boldsymbol{q}_k$ is the initial approximation of the solution, and $\boldsymbol{J}_r$ is the $m \times n$ Jacobian matrix, whose computation is described in detail in [28]. Let $\boldsymbol{e}_r = \boldsymbol{r} - f_r(\boldsymbol{q}) = f_r(\boldsymbol{q}_k)$ and $\Delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_k$, then from equation (5.1) we obtain

$$\boldsymbol{e}_r = \boldsymbol{J}_r(\boldsymbol{q}_k)\Delta\boldsymbol{q}, \tag{5.2}$$

Consider the optimization problem

$$\begin{aligned} \underset{\Delta\boldsymbol{q}\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\|\Delta\boldsymbol{q} + \boldsymbol{e}_q\|^2_{\boldsymbol{H}} \\ \text{subject to} \quad & \boldsymbol{e}_r = \boldsymbol{J}_r(\boldsymbol{q}_k)\Delta\boldsymbol{q}, \end{aligned} \tag{5.3}$$

where $\boldsymbol{e}_q = \mu_{ik}(\boldsymbol{q}_k - \boldsymbol{q}_0)$ is the weighted difference between the initial guess and the reference joint angles $\boldsymbol{q}_0$. $\boldsymbol{e}_q$ is added to the objective function in order to obtain repetitive solutions.

The optimization problem (5.3) is equivalent to

$$\begin{aligned} \underset{\Delta \boldsymbol{q} \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\|\Delta \boldsymbol{q}\|^2_{\boldsymbol{H}} + \boldsymbol{e}_q^T \boldsymbol{H} \Delta \boldsymbol{q} \\ \text{subject to} \quad & \boldsymbol{e}_r = \boldsymbol{J}_r(\boldsymbol{q}_k)\Delta \boldsymbol{q}. \end{aligned} \tag{5.4}$$

5.3.2 Implementation

The support foot is selected as the fixed base of the kinematic tree. Consequently, the base must be periodically changed. This change is performed in the end of **DS** and is a source of error, that must be accounted for (see Section Section 5.4.2). In theory, it is possible to always use the same foot as the base assuming that the position tracking for this foot is perfect, but this assumption is admissible only in simulation.

Alternatively, the torso can be selected as the base [14]. Due to implicit error feedback it might a better choice when a robot is subject to disturbances.

The functions that solve **FK**, compute the Jacobian $\boldsymbol{J}_r$ and $\boldsymbol{e}_r$ were generated using `Maple`[2].

Two **FK** functions are used. The first one determines pose of a swing foot, when change of the base is performed. The second computes the **CoM** position for error feedback.

The **QP** (5.4) is solved using block elimination with $\boldsymbol{H} = \boldsymbol{I}$, and μ_{ik} was determined experimentally. The problem is resolved until the change in joint angles becomes smaller than the minimal angle detected by the joint position sensors. Also, a limit on the number of iterations is imposed to detect cases, when there is no solution.

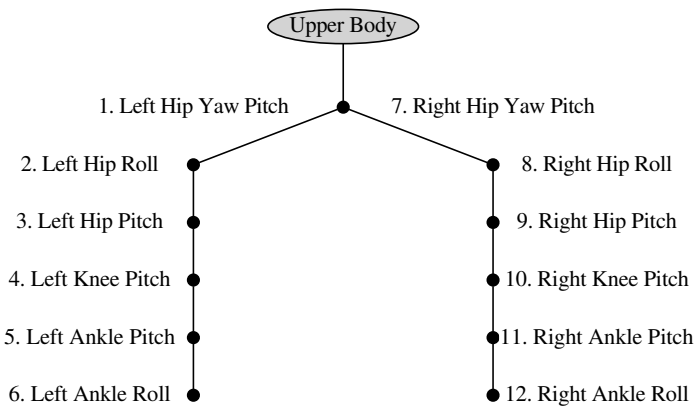


Figure 5.4: Lower body joints of Nao

[2]The code is available as well at `https://github.com/asherikov/`.

The joint angles of the upper body are fixed, hence only 12 lower body joints depicted in Figure 5.4 are controlled. Note that joints 1 and 7 are coupled [3], that is, they share the same motor and move simultaneously and symmetrically. In the case of conflicting orders `LHipYawPitch` always takes priority. The coupling between these joints is handled by imposing an equality constraint $\Delta \boldsymbol{q}_1 = \Delta \boldsymbol{q}_7$. It is also possible to reduce the number of controlled joints to account for this.

The total number of imposed constraints is 10:

- 3 on the position of the swing foot;
- 3 on the orientation of the swing foot;
- 3 on the position of the **CoM**;
- 1 to take into account the coupled joint.

Several alternative versions of **IK** implementation were also tested. In the early variants of the **IK** module the difference between the joint angles and the reference angles is not penalized, but the constraints on the orientation of the frame fixed to the **CoM** are used instead. These constraints are imposed in such a way, that the rotation about x and y axes is not allowed. The constraint on rotation about x axis was removed in order to reduce the load on the `HipRoll` joints, which tend to violate their limits for some **CoM** trajectories otherwise. Further experiments demonstrated, that the gait is better, when there are no constraints on the orientation of the **CoM**. Also, it is unclear how to define the desired orientation of the **CoM** in a general case, for example, during a circular walk. The penalization of the difference with the reference joint angles was introduced to maintain upright posture and avoid failures due to bad configurations of the lower body joints.

Another version of **IK** is similar to the current one, but it controls all joints of the robot instead of only the lower body joints. However, the ability to cope with external disturbances is roughly the same. Furthermore, this approach requires more time for computations.

5.4 Control Thread

The logic of one control loop of the control thread is shown in Figure 5.5.

5.4.1 Real-time Control Features of the Nao API

The control of a walking robot must be performed in real-time. **Application Programming Interface** (**API**) of a Nao robots provides two possible ways to implement this. The first one is to register callback function from **Device Communication Manager** (**DCM**), which is activated each 10 milliseconds. The main

[3] Refer to `http://www.aldebaran-robotics.com/documentation/index.html` for more information.

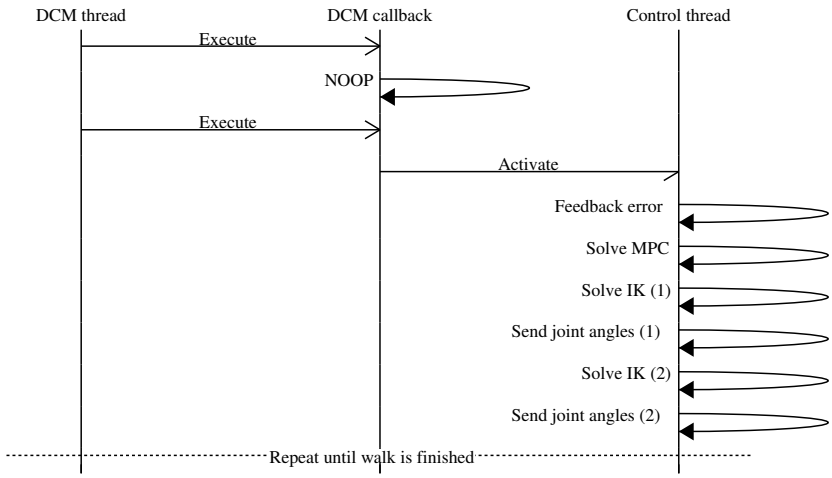


Figure 5.5: Control loop of the walking module

purpose of the **DCM** is to update sensor readings in the memory, since there is no direct access to the sensors. The second way is to register callback function from the built-in `Motion` module, which is activated each 20 milliseconds. The `Motion` module can perform walk and other operations, but it is activated even if no commands were sent.

We have tried to use both methods for real-time control. Experiments have demonstrated, that period of 10 milliseconds is too short to perform all necessary operations. On the other hand, when the callback from `Motion` module was used, it was observed, that the sensor readings are not necessary updated twice during the 20 milliseconds between activations of the module (at least in the simulation on a personal computer). Hence, a control thread was introduced, which is activated by a callback function that is called when **DCM** finishes its work. Thus the callback function is executed each 10 milliseconds. It has a counter of executions, and on each even execution (that is, each 20 milliseconds) it wakes up the control thread using a synchronization tool provided by the `Boost` library.

The priority of the control thread plays a critical role, since sometimes it is not possible to complete all necessary computations in available time using the default priority. We use `SCHED_FIFO` scheduling policy, the same as **DCM** thread uses. The priority is set to 65. Note that `NaoQi` does not run with root privileges and priority cannot be set arbitrarily.

5.4.2 Error Feedback

On the feedback step the position of the **CoM** is computed based on the current joint angles. Then the error between the computed and the expected positions of the **CoM** is computed. The expected position is taken from the previous solution of **MPCWMG**. If the error is below a given threshold, it is silently discarded, otherwise it is scaled by a gain and added to the expected position of the **CoM**. Error feedback for the velocity and acceleration of the **CoM** is not performed. The state obtained after error feedback is used as initial state of the **MPCWMG**.

There is also a second type of error feedback, which is performed when the reference foot is changed. Note that when the new reference foot touches the ground its position is not exactly the same as the target one. Therefore, if the reference frame is moved to this foot, an error in position of **CoM** appears. To avoid this effect the footstep pattern generator is informed about the error in foot position before the reference foot is changed. It is also necessary to synchronize joint angles in the model of the robot, which is used in inverse kinematics library, with the joint angles obtained from sensors.

5.4.3 Accounting for the Computational Delay

Quadratic programming is a time demanding task, and it takes a considerable part of one control sampling period. It is impossible to obtain smooth joint motions without accounting for the computational delay. The model of the system can be modified for this purpose as described in Section 2.5 of [16]. However, for the sake of simplicity the system was not modified, and a heuristic described in this section was used instead.

The simplest workaround is to continue execution of the commands obtained on the previous iteration until the new commands are available. On the other hand, if the old commands are not finished before the start of the next iteration, it might be difficult to perform error feedback, since it is necessary to have a good estimation of the desired state. The last problem can be addressed by solving the **IK** twice, so that there are two sets of commands: one must be executed in one control sampling period and the second one is used, while the new solution is not available. The joint angles from the second set are expected to be reached in two control sampling periods. As soon as the first solution of inverse kinematics is obtained the commands are sent to the joint controllers to replace the commands sent on the previous iteration. It takes 1 millisecond for joint controllers to react to the new commands. Note that the second **IK** solution can be used as initial guess for the first **IK** in the next control loop leading to a smaller number of **IK** iterations.

In some situations the computation time may exceed the time available for one iteration of the control loop, such situations are potentially dangerous and must be avoided. In order to enforce the time limits, a simple mechanism was

developed that interrupts the execution of the walk, if the duration of one iteration of the control loop exceeded 15 milliseconds.

Chapter 6
Experimental Results

This chapter describes the results of the experiments, which were conducted on a robot and in a simulation on a personal computer. The experiments on a robot were performed in order to evaluate and tune the control scheme as a whole. The comparison of the active set and logarithmic barrier solvers was also performed in simulation, since there are too many factors that can affect the performance on a robot, and it is rather difficult to apply disturbances to a robot in a repeatable manner. Note that even in the simulation it is necessary to check the feasibility of the generated **CoM** trajectory using the inverse kinematics module.

The results of the tests presented in this chapter were obtained during a straight walk. However, the walking module can realize more complex patterns, for example, circular walk[1].

6.1 Parameters of the Walk and Quadratic Problem

The walk is affected by a number of parameters. Some of them were preset, while others were tuned during experiments on a robot.

The step height and length were set beforehand. The height of the step is 2 centimeter. The step length is controlled by a parameter, which specifies the displacement of a footstep with respect to the previous footstep along the x axis. This parameter is equal to 4 centimeter, hence, the full step length is 8 centimeter.

The parameters of Bezier curves that are used to generate foot trajectories were found using trial and error approach. The trajectory is shown in Figure 5.3.

The length of one **SS** phase is set to 400 millisecond. The **DS** phase takes 120 millisecond and is approximated by 3 "fake" **DS** with rectangular constraints as described in Section 3.2.

[1]Video recordings of walk patterns realized by our control module can be accessed at `http://www.youtube.com/playlist?list=PL93B16B2910EC7F3D&feature=plcp`

The preview window includes 40 sampling intervals. The first two sampling intervals are set to 20 millisecond, since we use the first two states from the solution of **MPCWMG** (see Section 5.4.3 for more information). The default time for the remaining sampling intervals is 40 millisecond in order to obtain a longer preview window as explained in Section 3.3. However, sometimes it is not possible to set all remaining intervals to 40 millisecond and one of them must be also set to 20 millisecond. For example, if one iteration of the control loop is spent in **SS**, the remaining time in this **SS** is 380 millisecond, which cannot be divided into intervals of 40 millisecond. Thus, the total length of the preview window is $2 * 20 + 38 * 40 = 1560$ or $2 * 20 + 37 * 40 + 20 = 1540$ millisecond.

It is possible to increase the default length of a preview interval, but it has negative effects on the gait. Also, the length of all sampling intervals in the preview window can be set to 20 millisecond. In this case if the number of sampling intervals is not changed, the length of a preview window is significantly shorter and a robot does not compensate for external disturbances as well. On the other hand, if the number of sampling intervals is increased, a solution of a bigger **QP** problem takes more time.

The parameter μ_{ik}, which penalizes the difference with the reference joint angles in the inverse kinematics **QP** (Section 5.3.1), is set to 1.2.

The gain and threshold for error feedback of **CoM** position are set to 0.3 and 4 millimeters. The purpose of these parameters is described in Section 5.4.2. If the threshold is high, the compensation for external disturbances is not sufficient.

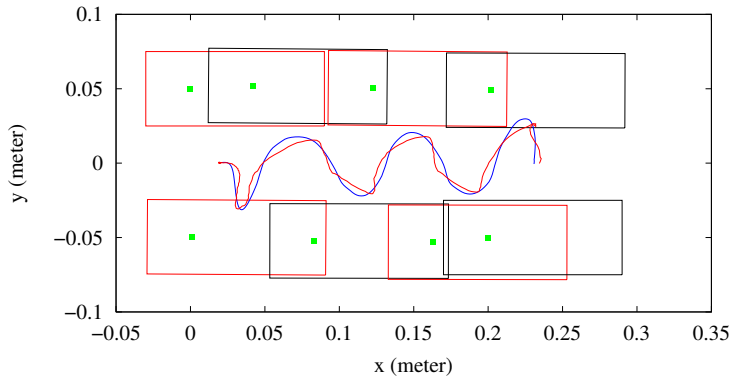
The solutions of **MPCWMG** are affected by four gains. The first one α_g penalizes the difference between the computed **ZMP** trajectory and reference **ZMP** points. The last three $\beta_g, \gamma_g, \eta_g$ penalize the absolute values of velocity, acceleration, and jerk of the **CoM**. It was observed, that α_g is the most important. The values of the gains are $\alpha_g = 8000$, $\beta_g = \gamma_g = \eta_g = 1$. If α_g is set to a smaller value, the threshold for **CoM** error feedback must be increased to avoid generation of infeasible **CoM** trajectories.

6.2 Experiments on a Robot

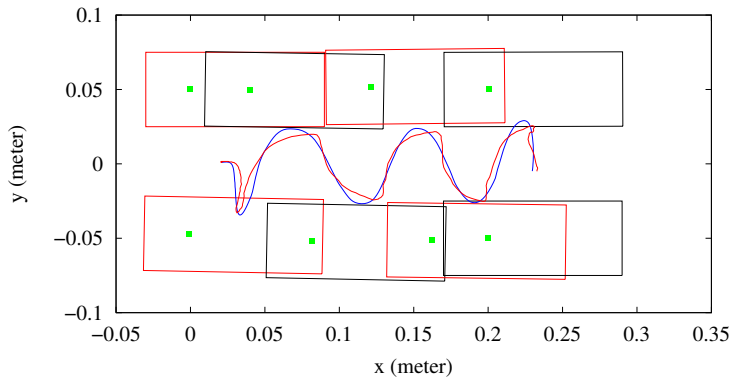
The purpose of the experiments conducted on a Nao robot was to evaluate quality of the produced gait, compare the two developed **MPCWMG** solvers, and identify potential improvements of the control scheme.

Initial tuning of the parameters of the walking module was performed using the active set method. Then the parameters of the logarithmic barrier method were found. The approximate solution of the **QP** by logarithmic barrier method does not have negative effects on the gait (see Figures 6.1 and 6.6).

During the straight walk the active set method does not deactivate constraints, the maximal number of added constraints is 7. In the initial and final



(a) **MPCWMG** is solved using the active set method



(b) **MPCWMG** is solved using the logarithmic barrier method

Figure 6.1: The footsteps and projection of **CoM** to x-y plane. The data is gathered during execution of a walking module on a robot. Red and black rectangles show positions of the footsteps. **CoM** trajectory generated by the solver is represented by the blue curve. The red line shows the measured position of **CoM**.

double supports constraints are not added, during the walk the number of activated constraints is usually 6 or 7. The parameters of the logarithmic barrier method were selected in such a way that it makes only one iteration of the external loop, then the number of internal loops was limited to 3.

Measurements of the execution time are presented in Table 6.1 and in Figure 6.2. The total execution time includes logging, which takes about 0.0005 second, solution of the **MPCWMG** problem, and solution of two inverse kinematics problems. The solution of one inverse kinematics problem takes from 1 to 4 iterations, in 80% of cases the number of iterations is not bigger than 2.

The generated foot trajectories are not followed precisely, due to the action of gravity on the swing foot, see Figure 6.3. Note that the right foot does not

	Total time (second)		Solution of the **QP** (second)	
	Active set method	Logarithmic barrier method	Active set method	Logarithmic barrier method
minimal	0.001867	0.004108	0.000490	0.002569
mean	0.003071	0.004834	0.001259	0.003016
maximal	0.004650	0.006012	0.002782	0.004126

Table 6.1: The amount of time spent in one iteration of the control loop and fraction of this time consumed by the solvers.

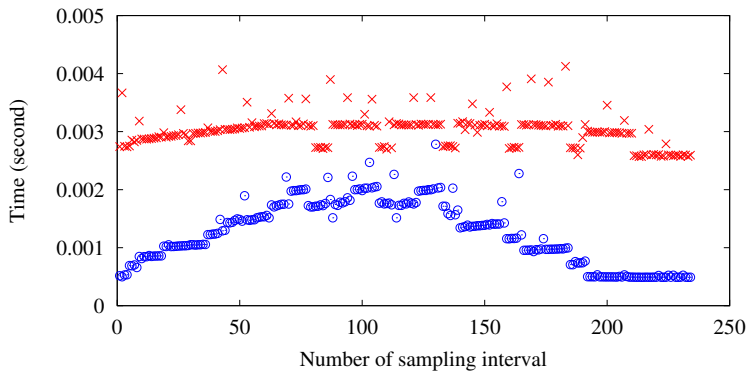


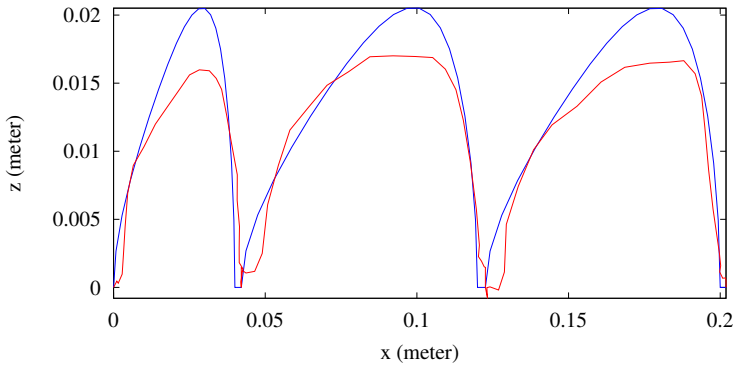
Figure 6.2: Execution time of the solvers on a robot during a straight walk. The red crosses show execution time of the logarithmic barrier method, the blue circles – the active set method.

track the desired trajectory as well as the left. This effect persists, if the walk is started from different leg. One of the possible reasons is a flaw of the hardware of the robot, which was used for tests.

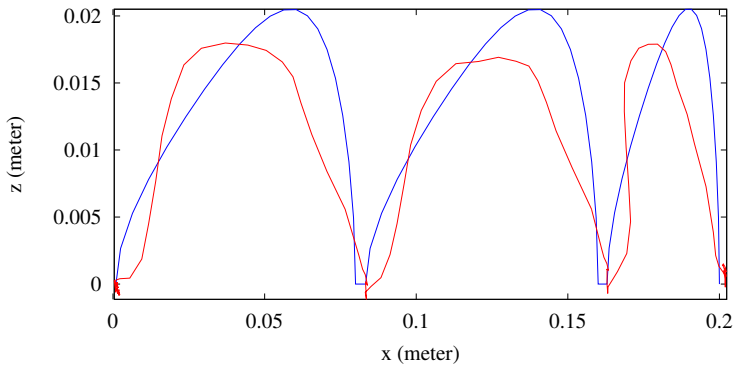
There is also an error in tracking of joint trajectories, an example is given in Figure 6.4. Several factors cause this error: action of the gravity, infeasible joint velocities, and the delay in sensor readings. It is desirable to account for some of these factors to improve the performance of the walking module.

Due to the errors in tracking of foot and joint trajectories and, probably, other factors, a significant error in **CoM** appears (see Figures 6.1, 6.5 and 6.6). The performance heavily depends on the error feedback. If the error feedback is disabled by setting the respective gain to zero, the robot starts to wobble after a few steps and soon falls.

It was mentioned in Section 5.4.2, that there is a second type of error feedback, which corrects the positions of footsteps. The result of this feedback can be clearly seen in Figure 6.1, when compared to the footsteps obtained in simulation that are shown in Figure 5.2.



(a) Trajectory of the left foot



(b) Trajectory of the right foot

Figure 6.3: Tracking of the foot trajectories in x-z plane. The blue line is the trajectory generated using Bezier curve, the red one shows the position of the foot computed by forward kinematics function based on the measured joint angles.

6.3 Experiments in a Simulation

One of the goals of the thesis is to compare active set and logarithmic barrier methods for solution of **MPCWMG**. Comparison of these methods in the presence of external disturbances on a robot is complicated due to the necessity to apply such disturbances in a repeatable manner. Hence, some of the tests were performed during simulation of the walk on a personal computer. The **CoM** trajectories were generated with the same parameters as on the robot. The inverse kinematics module was adopted to ensure feasibility of these trajectories, since neither optimality or stability of **MPCWMG** solutions imply feasibility (see Figure 6.8).

The **CoM** trajectories that were obtained in the simulation without any disturbances are shown in Figure 6.7.

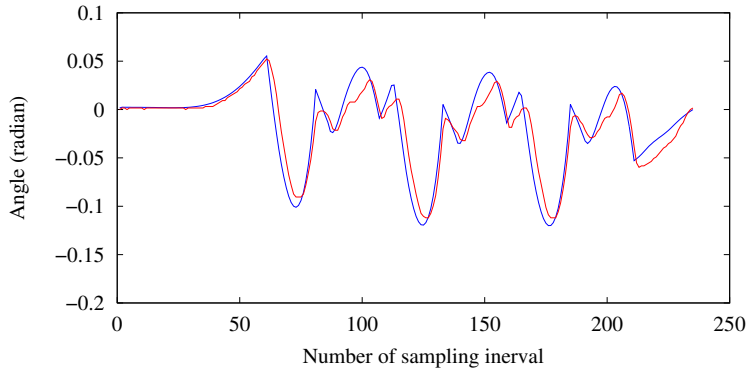


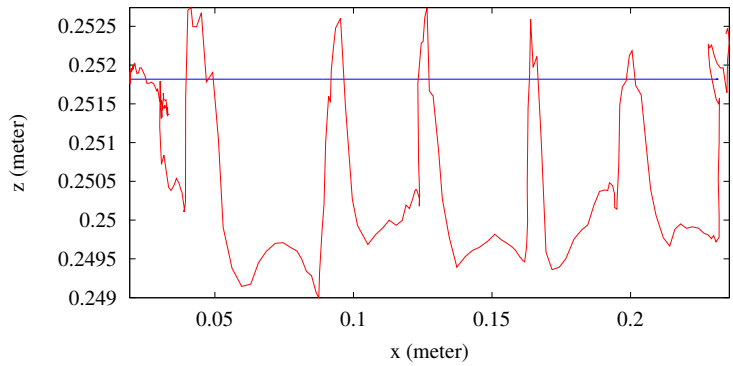
Figure 6.4: Tracking of the right hip roll joint trajectory produced by the inverse kinematics module. The blue curve is the generated trajectory, while the red one shows the measured joint angle.

The disturbances are applied by instantaneous change of the position of **CoM** to replicate error feedback on the robot. An example is shown in Figure 6.8. During one simulation run the system is disturbed only once. The displacement of **CoM** is no more than two centimeters along one of the axes in the horizontal plane.

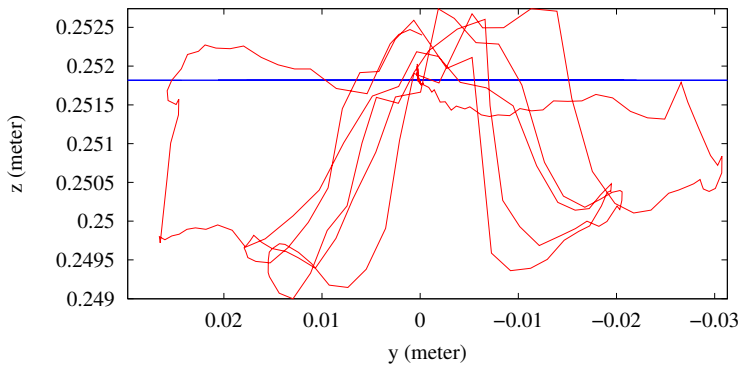
It was observed that strong disturbances (more than one centimeter) often lead to instability of the **MPCWMG**, when the **CoM** trajectory diverges.

The disturbances have a strong effect on the performance of the active set method, since the number of constraints at the optimal point is often higher. In some situations the size of the active set reaches 40, which makes the solution computationally infeasible on a robot. In order to enforce the time limits, the deactivation of constraints can be disabled and the maximal number of activated constraints can be fixed. If the limit on the number of the activated constraints is relatively high (bigger than 20), the approximate solution is often admissible. However, strict limits, as well as disabled deactivation of constraints may lead to instability.

The **MPCWMG** may also become unstable, if there are no disturbances, but a rather strict limit on the number of active constraints is imposed. The properly tuned logarithmic barrier method can be safer in this case, since it results in higher decrease rate of the value of the objective function (see Figure 6.9) in the number of iterations.

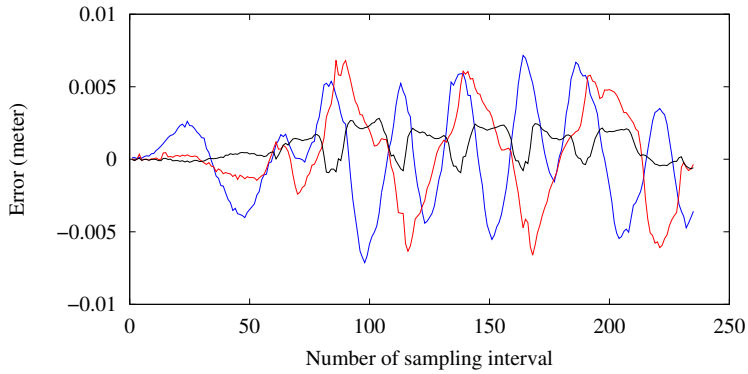


(a) Projection of **CoM** trajectory to x-z plane

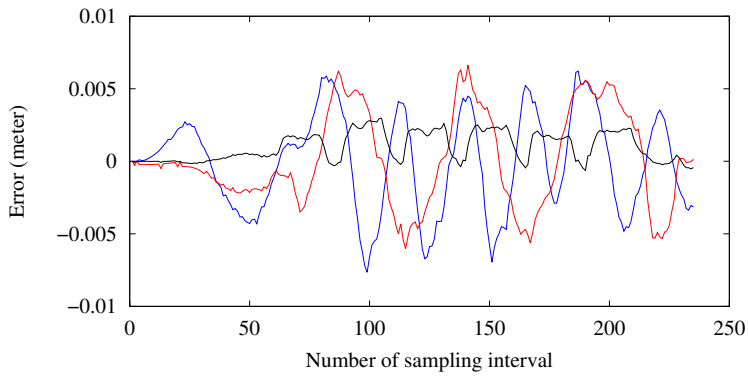


(b) Projection of **CoM** trajectory to y-z plane

Figure 6.5: Projections of **CoM** trajectory generated using the active set method to x-z and y-z planes. The data is gathered during execution of a walking module on a robot. The reference height of **CoM**, which is always constant, is represented by the blue line. The red curve shows the measured position of **CoM**.

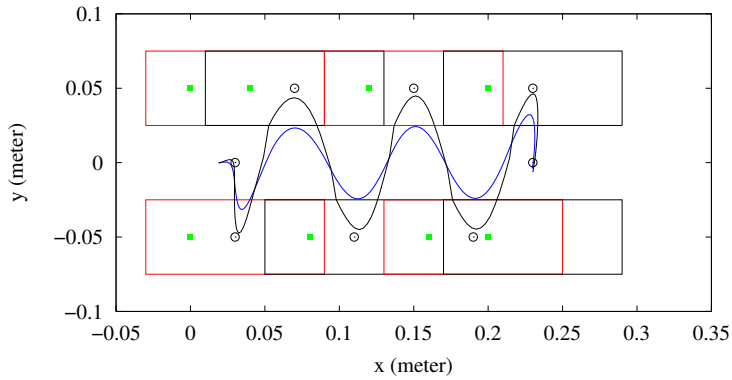


(a) **MPCWMG** is solved using the active set method

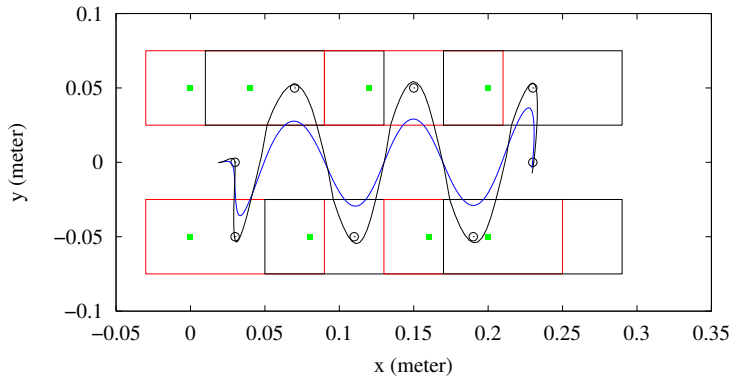


(b) **MPCWMG** is solved using the logarithmic barrier method

Figure 6.6: Error in **CoM** position. Blue, red, and black lines correspond to error along x, y, and z axes, respectively.

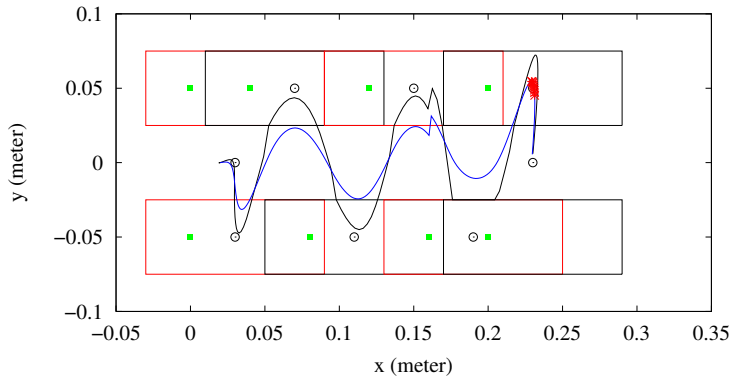


(a) Trajectory generated using the active set method

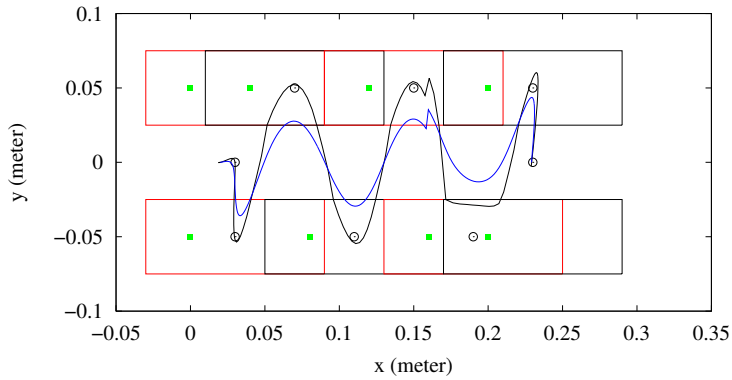


(b) Trajectory generated using the logarithmic barrier method

Figure 6.7: Trajectories of **CoM** in a simulation. Red and black rectangles represent **SS**, **DS** are omitted. Blue and black curves show trajectories of **CoM** and **ZMP**, respectively. Black circles stand for reference **ZMP** positions. Green boxes are the footstep reference points.



(a) Trajectory generated using the active set method



(b) Trajectory generated using the logarithmic barrier method

Figure 6.8: Trajectories of **CoM** in the presence of disturbance. Red and black rectangles represent **SS**, **DS** are omitted. Blue and black curves show trajectories of **CoM** and **ZMP**, respectively. Black circles stand for reference **ZMP** positions. Green boxes are the footstep reference points. The red stars indicate infeasible **CoM** positions.

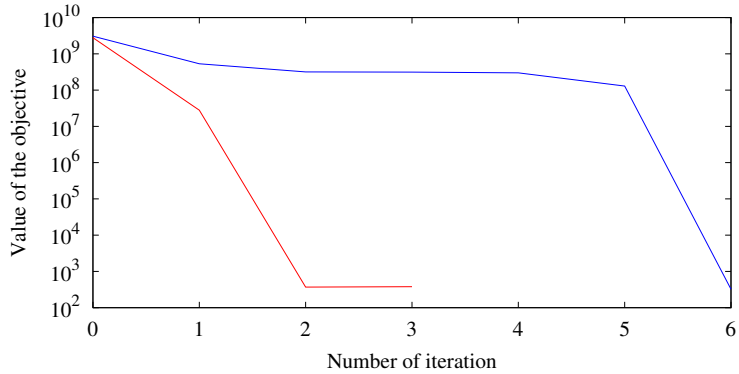


Figure 6.9: The decrease rate of objective function. The values produced using active set and logarithmic barrier method are represented by the blue and red lines, respectively.

Chapter 7
Conclusion

A short summary of the work presented in the thesis is given in this chapter. This work includes development of walking control scheme, its tuning and testing, and discussion of associated issues.

Several software components were developed in order to test the control scheme on a robot:

- the sparse active set and logarithmic barrier solvers for **MPCWMG** (Chapters 4 and 5);
- the inverse kinematics library (Chapter 5);
- the footstep pattern generator (Chapter 5);
- the middleware (Chapter 5), which integrates other components and interacts with the Nao robot.

In order to obtain an admissible gait, a number of experiments were conducted using the walking module (Chapter 6). The experiments allowed to select an appropriate version of inverse kinematics, find good trajectories for the swing foot, and determine parameters of the **MPCWMG** as well as parameters of the solvers.

The performance of the active set solver based on the sparse formulation compared to the dense formulation was presented in [9]. Though, the sparse formulation may improve performance, its implementation and extension (for example, to perform footstep repositioning [8]) requires more effort.

The sparse active set and logarithmic barrier solvers were compared on a robot and in a simulation under disturbances (Chapter 6). The results demonstrated, that the logarithmic barrier method is sufficiently fast for the real-time operation on a robot, and may be preferable in some situations.

Some heuristics were also tested:

- Approximation of the **DS** with a sequence of rectangles (Section 3.2) works reasonably well, however, the narrower rectangular constraints may lead to a higher number of active constraints at the optimal point.

- Variation of the duration of the sampling periods in the preview window (Sections 3.3 and 6.1) gives more freedom for tuning, but distribution of the **DS** and **SS** between different sampling periods is problematic.
- Another heuristic is employed to account for the computational delay of the control loop (Section 5.4.3).

The rest of the chapter is devoted to discussion of the possible future work.

7.1 Discussion of Future Work

The possible extensions of the work presented in the thesis can be divided into two groups. The first group is focused on the control scheme, while the second one on its implementation.

The control scheme can be potentially improved in several ways. It is interesting to investigate the effect of introduction of stabilizing constraints to the **MPCWMG** as described in Section 3.5. The joint limits can be explicitly accounted for in the **IK** problem (see Section 5.3.1). However, the **IK** problem with the inequality constraints would need a **QP** solver. The **IK** module can be rewritten in order to take the position of torso as the base in the same way as it is made in [14]. The precision of the **IK** may be improved by exclusion of the support foot/feet from computation of the position of the **CoM**. Note that the sparse **MPCWMG** formulation allows easy change of the height of **CoM** during walk. Exploitation of this feature may lead to a more natural gait. It is also possible to implement and test on a robot repositioning of footsteps under disturbances, which is described in [8].

The second group of extensions include various heuristics and fine adaption of the walking module to the Nao platform. The available pressure sensors can be used to improve error feedback. The module can be modified to account for the sensor delay, which is quite significant (10 millisecond). Inclination of the sole of the swing foot may improve walk on uneven terrain.

Appendix A
Derivation of Schur Complement

Here an example derivation of the Schur complement of the **KKT** system (4.4) is presented.

The Schur complement in Section 4.3.2 was derived as

$$\boldsymbol{S} = \frac{1}{2}\tilde{\boldsymbol{E}}_c \tilde{\boldsymbol{H}}_c^{-1} \tilde{\boldsymbol{E}}_c^T + \frac{1}{2}\tilde{\boldsymbol{E}}_u \boldsymbol{H}_u^{-1} \tilde{\boldsymbol{E}}_u^T.$$

When $N = 3$ the two terms in the summation are

$$\begin{aligned}
\tilde{\boldsymbol{E}}_c \tilde{\boldsymbol{H}}_c^{-1} \tilde{\boldsymbol{E}}_c^T &= \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \tilde{\boldsymbol{A}}_1 & -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{A}}_2 & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{Q}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{Q}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{Q}} \end{bmatrix}^{-1} \begin{bmatrix} -\boldsymbol{I} & \tilde{\boldsymbol{A}}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} & \tilde{\boldsymbol{A}}_2^T \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{Q}}^{-1} & -\tilde{\boldsymbol{Q}}^{-1}\tilde{\boldsymbol{A}}_1^T & \boldsymbol{0} \\ -\tilde{\boldsymbol{A}}_1\tilde{\boldsymbol{Q}}^{-1} & \tilde{\boldsymbol{A}}_1\tilde{\boldsymbol{Q}}^{-1}\tilde{\boldsymbol{A}}_1^T + \tilde{\boldsymbol{Q}}^{-1} & -\tilde{\boldsymbol{Q}}^{-1}\tilde{\boldsymbol{A}}_2^T \\ \boldsymbol{0} & -\tilde{\boldsymbol{A}}_2\tilde{\boldsymbol{Q}}^{-1} & \tilde{\boldsymbol{A}}_2\tilde{\boldsymbol{Q}}^{-1}\tilde{\boldsymbol{A}}_2^T + \tilde{\boldsymbol{Q}}^{-1} \end{bmatrix}, \\
\tilde{\boldsymbol{E}}_u \boldsymbol{H}_u^{-1} \tilde{\boldsymbol{E}}_u^T &= \begin{bmatrix} \tilde{\boldsymbol{B}}_0 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{B}}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{B}}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{P} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P} \end{bmatrix}^{-1} \begin{bmatrix} \tilde{\boldsymbol{B}}_0^T & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{B}}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{B}}_2^T \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{P}}_0 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{P}}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{P}}_2 \end{bmatrix},
\end{aligned}$$

where $\tilde{\boldsymbol{P}}_k = \tilde{\boldsymbol{B}}_k \boldsymbol{P}^{-1} \tilde{\boldsymbol{B}}_k^T$.

Hence, $\boldsymbol{S}$ has a block-diagonal structure with the following blocks

$$\begin{aligned} 2\boldsymbol{S}_{11} &= \tilde{\boldsymbol{Q}}^{-1} + \tilde{\boldsymbol{P}}_0 \\ 2\boldsymbol{S}_{kk} &= \boldsymbol{A}_{k-1}\tilde{\boldsymbol{Q}}^{-1}\boldsymbol{A}_{k-1}^T + \tilde{\boldsymbol{Q}}^{-1} + \tilde{\boldsymbol{P}}_{k-1} \\ 2\boldsymbol{S}_{k,k+1} &= \boldsymbol{S}_{k+1,k}^T = -\tilde{\boldsymbol{Q}}^{-1}\boldsymbol{A}_k^T. \end{aligned}$$

References

[1] R.A. Bartlett, A. Wachter, and L.T. Biegler. Active set vs. interior point strategies for model predictive control. In *American Control Conference, 2000. Proceedings of the 2000*, volume 6, pages 4229 –4233 vol.6, 2000. (Cited on page 17.)

[2] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3 – 20, 2002. (Cited on page 10.)

[3] H. G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984. (Cited on page 18.)

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. (Cited on pages 17 and 26.)

[5] T. Bretl and S. Lall. Testing static equilibrium for legged robots. *Robotics, IEEE Transactions on*, 24(4):794 –807, aug. 2008. (Cited on page 5.)

[6] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on Passive-Dynamic walkers. *Science*, 307(5712):1082–1085, 2005. (Cited on page 4.)

[7] Moritz Diehl, HansJoachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In Lalo Magni, DavideMartino Raimondo, and Frank Allgwer, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 391–417. Springer Berlin Heidelberg, 2009. (Cited on page 18.)

[8] D. Dimitrov, A. Paolillo, and P.B. Wieber. Walking motion generation with online foot position adaptation based on ℓ_1-and ℓ_∞-norm penalty

formulations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3523–3529. IEEE, 2011. (Cited on pages 10, 53, and 54.)

[9] D. Dimitrov, A. Sherikov, and P.B. Wieber. A sparse model predictive control formulation for walking motion generation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2292–2299. IEEE, 2011. (Cited on pages 9, 18, and 53.)

[10] D. Dimitrov, P.B. Wieber, H.J. Ferreau, and M. Diehl. On the implementation of model predictive control for on-line walking pattern generation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2685–2690. IEEE, 2008. (Cited on page 13.)

[11] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008. (Cited on page 18.)

[12] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A schur-complement method for sparse quadratic programming. Technical report, Stanford University, 1987. (Cited on page 21.)

[13] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996. (Cited on pages 22 and 24.)

[14] D. Gouaillier, C. Collette, and C. Kilner. Omni-directional closed-loop walk for nao. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 448 –454, dec. 2010. (Cited on pages 8, 15, 32, 35, and 54.)

[15] H. Hirukawa, S. Hattori, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara, and M. Morisawa. A universal stability criterion of the foot contact of legged robots - adios zmp. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1976 –1983, may 2006. (Cited on page 5.)

[16] Maciejowski J.M. *Predictive Control with Constraints*. Prentice-Hall, 2002. (Cited on pages 9 and 38.)

[17] S. Kajita and B. Espiau. Legged robots. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 361–389. Springer Berlin Heidelberg, 2008. (Cited on pages 4 and 6.)

[18] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. volume 2, pages 1620 – 1626, sept. 2003. (Cited on pages 6, 9, and 15.)

[19] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa. A realtime pattern generator for biped walking. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 31 – 37 vol.1, 2002. (Cited on page 6.)

[20] C.C. Kemp, P. Fitzpatrick, H. Hirukawa, K. Yokoi, K. Harada, and Y. Matsumoto. Humanoids. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 361–389. Springer Berlin Heidelberg, 2008. (Cited on page 1.)

[21] J. Liu, F. Xue, and X. Chen. A universal biped walking generator for complex environments with pattern feasibility checking. *International Journal of Humanoid Robotics*, 8(2):323–357, 2011. (Cited on page 7.)

[22] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789 – 814, 2000. (Cited on page 14.)

[23] Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990. (Cited on page 4.)

[24] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara, S. Nakaoka, and H. Hirukawa. A biped pattern generation allowing immediate modification of foot placement in real-time. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 581 – 586, dec. 2006. (Cited on page 7.)

[25] Kenneth R. Muske and James B. Rawlings. Model predictive control with linear models. *AIChE Journal*, 39(2):262–287, 1993. (Cited on page 15.)

[26] K. Nishiwaki and S. Kagami. Online walking control system for humanoids with short cycle pattern generation. *The International Journal of Robotics Research*, 28(6):729–742, 2009. (Cited on page 17.)

[27] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, second edition, 2006. (Cited on pages 17 and 19.)

[28] A. Paolillo. Online walking motion generation for a humanoid robot based on model predictive control. Master's thesis, Sapienza University of Rome, 2010. (Cited on pages 1, 2, and 34.)

[29] C. Y. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *J. Optim. Theory Appl.*, 99(3):723–757, December 1998. (Cited on pages 17 and 18.)

[30] M. Vukobratović and B. Borovać. Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004. (Cited on page 5.)

[31] M. Vukobratović, B. Borovac, and V. Potkonjak. Towards a unified understanding of basic notions and terms in humanoid robotics. *Robotica*, 25(1):87–101, January 2007. (Cited on pages 4 and 5.)

[32] Yang Wang and S. Boyd. Fast model predictive control using online optimization. *Control Systems Technology, IEEE Transactions on*, 18(2):267 –278, march 2010. (Cited on page 18.)

[33] Pierre-Brice Wieber. On the stability of walking systems. In *Proceedings of the International Workshop on Humanoid and Human Friendly Robotics*, Tsukuba, Japon, 2002. (Cited on page 5.)

[34] Pierre-Brice Wieber. Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations. In *IEEE-RAS International Conference on Humanoid Robots*, Genova, Italie, 2006. (Cited on pages 9 and 10.)

[35] Yunong Zhang, Zhiguo Tan, Ke Chen, Zhi Yang, and Xuanjiao Lv. Repetitive motion of redundant robots planned by three kinds of recurrent neural networks and illustrated with a four-link planar manipulator's straight-line example. *Robotics and Autonomous Systems*, 57(6–7):645 – 651, 2009. (Cited on page 34.)