A Model-Based Control Approach for Locomotion of Biped Robots

Program:
Robotics and Autonomous Systems

Student:
Salman Faraji

Supervisor:
Professor Auke Jan Ijspeert

Assistant:
Soha Pouya

2.8.2013
Lausanne, Switzerland
This project was started on 8.4.2013
Abstract

In this research we aim at proposing a general novel walking method for locomotion of torque controlled robots. The method should be able to produce a wide range of speeds without requiring off-line optimizations and re-tuning of parameters. It should be capable of tolerating internal errors, noises and control delays as well as external disturbances such as pushes or roughness in the environment. We have a quadratic whole-body optimization which generates joint torques, given desired Cartesian accelerations of center of mass and feet. Using dynamics model of the robot inside this optimizer ensures compliance and better tracking, required for fast locomotion. We have simplified the model of robot to linear inverted pendulum and proposed different planners which are other quadratic convex problems optimizing future behavior of the robot. These planners are in fact model predictive control which optimize the system either in continuous or discrete time domains. Fast libraries help us performing these calculations per time step and producing desired motion. With very few parameters to tune and no perception, our method shows notable robustness against strong external pushes, large terrain variations, internal noises, model errors and also delayed communication. Evident by various simulations in different conditions, we can suggest our general method for walking control of a wide range of humanoid robots.\footnote{Watch all movies for different simulations in this work at http://biorob.epfl.ch/page-96274.html}
# Contents

<table>
<thead>
<tr>
<th>List of figures</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of tables</td>
<td>ix</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Main questions to answer

1.1.1 Energy efficiency

1.1.2 Mechanical power

1.1.3 Compliance

1.1.4 Agility

1.1.5 Robustness

1.2 Contributions of this work

## 2 Literature review

2.1 Model-Free Methods

2.1.1 Central Pattern Generators (CPG)

2.1.2 Reflex based control

2.2 Kinematic-model based methods

2.2.1 Inverse kinematics

2.2.2 Zero Moment Point (ZMP)

2.2.3 Potential fields

2.2.4 Virtual Model Control (VMC)

2.3 Dynamic-model based methods

2.3.1 Inverse dynamics

2.3.2 Projected inverse dynamics

2.3.3 Trajectory optimization

2.3.4 Hybrid zero dynamics

2.3.5 Operational space control

2.3.6 Whole body optimization

2.4 Conclusion
# Simulation platform

## 3.1 Atlas robot

## 3.2 Simulation software
- 3.2.1 Gazebo
- 3.2.2 DRCSim
- 3.2.3 ROS
- 3.2.4 CloudSim

## 3.3 Control packages
- 3.3.1 SD/FAST
- 3.3.2 CVXGEN

## 3.4 Perception
- 3.4.1 Constraint consistent local position
- 3.4.2 Incremental odometry

# Whole body optimization, generating joint torques

## 4.1 Designing low level controller
- 4.1.1 On-board joint controller in simulator
- 4.1.2 Robot states, combined position/force control
- 4.1.3 Abstracting the model to center of mass
- 4.1.4 Swing foot accelerations
- 4.1.5 CoP, friction and joint torque constraints
  - CoP constraints
  - Friction constraints
  - Joint torque limits
  - Existence of constraint
- 4.1.6 Formulating final quadratic problem
- 4.1.7 Contact force distribution
- 4.1.8 Imposed state of the low level controller
- 4.1.9 Experiment setup
- 4.1.10 Posture control
- 4.1.11 CoM acceleration regularization
- 4.1.12 Overall structure of low level controller

## 4.2 Testing low level controller
- 4.2.1 Scenario 1: Squatting
- 4.2.2 Scenario 2: Sagittal motion
- 4.2.3 Scenario 3: Push recovery and compliance test
- 4.2.4 Scenario 4: Statically-Stable walking

## 4.3 Conclusion
5 Control Method 1: Captured walking

5.1 3D Linear Inverted Pendulum Model (LIPM)

5.1.1 LIPM with point ground contact
5.1.2 LIPM with support foot
5.1.3 LIPM with support foot and inertia mass
5.1.4 Instantaneous capture point

5.2 Captured walking using model predictive control

5.2.1 Navigator
5.2.2 Stance planner, model predictive control
5.2.3 Swing planner, LIPM accelerations
5.2.4 Parameter tuning

5.3 Simulation of walking with different speeds

5.4 conclusion

6 Control Method 2: Dynamic walking

6.1 Literature review on planning methods

6.2 Model predictive control: Step planning

6.2.1 Navigator design
6.2.2 Planner design
6.2.3 Parameter tuning

6.3 Simulation of walking

6.3.1 Speed variations
6.3.2 External pushes
6.3.3 Delayed communication
6.3.4 System noises
6.3.5 Model errors in link masses
6.3.6 Model error in link lengths
6.3.7 Walking on rough terrain
6.3.8 Walking on different slopes

6.4 Summary
## List of Figures

1.1 Different quadruped robots ........................................... 2

2.1 CPG method .......................................................... 11
2.2 Reflex muscles ..................................................... 12
2.3 ZMP definition ...................................................... 14
2.4 Obstacle avoidance .................................................. 16
2.5 VMC external guidance ............................................. 17
2.6 VMC state machine ................................................ 18
2.7 Virtual model control ............................................... 18
2.8 Zero dynamics trajectory space .................................... 26

3.1 Atlas CAD model ..................................................... 32
3.2 Virtual challenge environment ..................................... 34
3.3 ROS visualization ................................................... 35

4.1 Control structure .................................................... 41
4.2 2D inverted pendulum ............................................. 55
4.3 Low level controller architecture ................................ 57
4.4 Squatting robot ...................................................... 58
4.5 Squatting trajectories ............................................... 59
4.6 Left/Right moving robot ............................................ 59
4.7 Left/Right motion trajectories ..................................... 61
4.8 Left/Right motion, saturated accelerations ..................... 62
4.9 Pushed robot .......................................................... 63
4.10 Push trajectories .................................................... 64
4.11 Static walking state machine ..................................... 65
4.12 Static walking 3D foot sequence ................................. 68
4.13 Static walking, close feet .......................................... 68
4.14 Static walking, fast sagittal motion ............................... 69
4.15 Static walking, accelerations ..................................... 70

5.1 Point contact LIPM .................................................. 73
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>LIPM with foot</td>
<td>75</td>
</tr>
<tr>
<td>5.3</td>
<td>LIPM with inertia mass</td>
<td>76</td>
</tr>
<tr>
<td>5.4</td>
<td>Capture point behavior</td>
<td>78</td>
</tr>
<tr>
<td>5.5</td>
<td>Controller architecture, continuous time MPC</td>
<td>79</td>
</tr>
<tr>
<td>5.6</td>
<td>1-Step capturability</td>
<td>80</td>
</tr>
<tr>
<td>5.7</td>
<td>N-Step capturability</td>
<td>81</td>
</tr>
<tr>
<td>5.8</td>
<td>Captured walking, state machine</td>
<td>86</td>
</tr>
<tr>
<td>5.9</td>
<td>Captured walking, comparison of different speeds</td>
<td>89</td>
</tr>
<tr>
<td>5.10</td>
<td>Captured walking, 3D foot sequence</td>
<td>90</td>
</tr>
<tr>
<td>5.11</td>
<td>Captured walking, MPC tracking performance</td>
<td>91</td>
</tr>
<tr>
<td>5.12</td>
<td>Captured walking, fast forward acceleration</td>
<td>92</td>
</tr>
<tr>
<td>5.13</td>
<td>Captured walking, push recovery</td>
<td>93</td>
</tr>
<tr>
<td>5.14</td>
<td>Captured walking, discrete time MPC</td>
<td>99</td>
</tr>
<tr>
<td>6.1</td>
<td>Controller architecture, discrete time MPC</td>
<td>105</td>
</tr>
<tr>
<td>6.2</td>
<td>Dynamic walking, ideal step sequence</td>
<td>106</td>
</tr>
<tr>
<td>6.3</td>
<td>Dynamic walking, state machine</td>
<td>109</td>
</tr>
<tr>
<td>6.4</td>
<td>Dynamic walking, 3D foot sequence</td>
<td>109</td>
</tr>
<tr>
<td>6.5</td>
<td>Dynamic walking, CoM bouncing</td>
<td>109</td>
</tr>
<tr>
<td>6.6</td>
<td>Dynamic walking, desired speed tracking</td>
<td>110</td>
</tr>
<tr>
<td>6.7</td>
<td>Dynamic walking, 0.3m/s limit cycles</td>
<td>111</td>
</tr>
<tr>
<td>6.8</td>
<td>Dynamic walking, planned foot sequence</td>
<td>112</td>
</tr>
<tr>
<td>6.9</td>
<td>Dynamic walking, different desired speeds</td>
<td>114</td>
</tr>
<tr>
<td>6.10</td>
<td>Dynamic walking, push recovery in 0.0m/s</td>
<td>115</td>
</tr>
<tr>
<td>6.11</td>
<td>Dynamic walking, push recovery with collision avoidance</td>
<td>117</td>
</tr>
<tr>
<td>6.12</td>
<td>Dynamic walking, push recovery during walking at 0.4m/s</td>
<td>118</td>
</tr>
<tr>
<td>6.13</td>
<td>Communication delays</td>
<td>120</td>
</tr>
<tr>
<td>6.14</td>
<td>Dynamic walking, noisy system at 0.0m/s</td>
<td>122</td>
</tr>
<tr>
<td>6.15</td>
<td>Model errors in masses</td>
<td>123</td>
</tr>
<tr>
<td>6.16</td>
<td>Dynamic walking, limit cycles of systems with mass model errors</td>
<td>124</td>
</tr>
<tr>
<td>6.17</td>
<td>Model errors in lengths</td>
<td>125</td>
</tr>
<tr>
<td>6.18</td>
<td>Dynamic walking, limit cycles of systems with length model errors</td>
<td>126</td>
</tr>
<tr>
<td>6.19</td>
<td>Rough terrain picture</td>
<td>127</td>
</tr>
<tr>
<td>6.20</td>
<td>Dynamic walking on rough terrain</td>
<td>127</td>
</tr>
<tr>
<td>6.21</td>
<td>Sloped terrain picture</td>
<td>128</td>
</tr>
<tr>
<td>6.22</td>
<td>Dynamic walking on slopes</td>
<td>129</td>
</tr>
<tr>
<td>7.1</td>
<td>Overall controller architecture</td>
<td>132</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Low level controller parameters</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Orientation controller parameters</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Static walking parameters</td>
<td>67</td>
</tr>
<tr>
<td>5.1</td>
<td>Captured walking parameters</td>
<td>87</td>
</tr>
<tr>
<td>6.1</td>
<td>Dynamic walking parameters</td>
<td>108</td>
</tr>
<tr>
<td>6.2</td>
<td>Push recovery comparison</td>
<td>118</td>
</tr>
<tr>
<td>6.3</td>
<td>Maximum tolerable noise in dynamic walking</td>
<td>122</td>
</tr>
<tr>
<td>6.4</td>
<td>Maximum model error in dynamic walking</td>
<td>123</td>
</tr>
<tr>
<td>7.1</td>
<td>Comparison of static, captured and dynamic walking methods</td>
<td>133</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Although wheeled robots are widely used for different applications, legged robots have also their own advantages in locomotion. They can potentially perform much better on rough terrains and complex environments while wheeled robots perform better in terms of speed and energy efficiency. Legged and armed structures are being mostly inspired from animals, but with simpler actuation properties. However, the complexity of structure, considerable number of actuators and degrees of freedom has made control problem a great challenge for engineers.

Animals in nature perform locomotion very easily without taking care of the type of terrain or other unwanted factors like constant wind or slopes. However, they perform some higher level thinking while encountering sudden changes in environment like reaching some stairs or being pushed by wind gusts. It is known from biology that most of the rhythmic motions are generated in low level and task space movements corresponding to desired goals are planned in higher levels of thinking process. We will comment more on these concepts in next section.

From the viewpoint of control, it is desired to decouple these tasks and to control them separately. One might produce joint motions in low level while handle higher level planning, navigation or any other task space motion on top of that using various kinds of information from the environment. So, low level control of robot joints intrinsically needs a fast loop with simple kinds of sensors while high level thinking requires collecting and processing of more complicated information and thus, more complex intelligent and heavier computations.

1.1 Main questions to answer

The topic of this project is related to legged locomotion and we want to propose a method that can answer to many challenging questions in these kinds of robots. The most important concepts in robotics and specifically legged robotics are energy
1.1.1 Energy efficiency

Energy is one of the most important parameters to be minimized in various kinds of robots. It is more important in mobile robots as they are not connected to a power supply constantly and they have to carry their power source, either batteries or gasoline or even solar panels. While energy efficiency has mostly commercial importance in industrial or fixed-based robotics, i.e. to minimize energy costs, it has crucial effect on the mechanical properties of mobile robots. A legged or wheeled robot might lose agility if it has to carry a big power source or an aerial robot’s flight time might decrease considerably for the same reason. Although wheeled robots usually have better efficiencies, there are many tasks could be handled merely by legged robots and this leaves the space open for researches on minimizing energy consumption in legged robots. Of course this variable highly depends on mechanical properties and actuation types of robots which we discuss shortly later. However, from control point of view, one is interested in proposing better methods that consume less energy. In the area of legged robotics, Fig 1.1 shows various robots with different energy consumptions.

To quantitatively measure the energy consumption, a parameter called Cost Of Transportation (COT) defined as:

\[ \text{COT} = \frac{\text{Energy Consumption}}{\text{Distance Traveled}} \]
\[ COT = \frac{\text{Energy}}{\text{Weight} \times \text{Distance}} \] (1.1)

In Fig.1.1 a), the passive walker robot [1] is able to walk a slope down without consuming any energy (COT=0). The idea is promising, showing that natural dynamics of robots can produce walking. Based on this concept, the Cornell Ranger [2] was built by the group of Ruina as shown in Fig.1.1 b), achieving a new record of 65km ultramarathon walk without stop on single charge. This light-weight robot uses trajectory optimization and reflexes to stabilize and has COT=0.28 which is a bit more than human (COT=0.2). In the area of humanoid robotics, the well-known Assimo robot by Honda in Fig.1.1 c) consumes a lot of energy and shows a stiff walking (COT=2). After all, it is the BigDog robot [3] shown in Fig.1.1 d) by Boston Dynamics who consumes a lot of energy and basically, it is not an electrical robot like others. Energy consumption is one axis, however the capabilities of performing different tasks and locomotion between above-mentioned robots are never comparable. So the goal of robotics is to make more efficient robots to be as close to humans as possible, either proposing better hardware or more energy-efficient control algorithms.

1.1.2 Mechanical power

Engineers design robot hardwares regarding their given task. If an industrial robot has to lift very heavy items or any other hard task quickly, the solution is not electrical motors probably and people use hydraulic actuators more often. In mobile robots, the main task is to move or carry different things. Metrics involved in the design of a robot are total expected amount of energy, weight, and the rate of energy consumption, i.e. power. Based on these parameters, different categories of actuators are used:

- **Electric**: These actuators are in fact converting electrical power to motion. So usually a heavy weight is expected for either battery or solar panels. They are easy to control and commercially having reasonable price. In terms of power however, they are not so strong and usually not used when a robot is expected to perform agile and dexterous tasks.

- **Pneumatic**: These actuators work with air pressure and are usually more strong than electrical ones. Due to complicated dynamics of air-flow, these actuators are rather difficult to control and they usually need large facilities nearby to produce air-pressure (air tank and compressor). One should think of another energy source as well to empower these facilities. Actuators themselves are not so heavy however.
• **Hydraulic:** These actuators are the strongest one, commercially available now. Like previous category, they also need same facilities to produce oil-pressure. The BigDog robot we introduced before uses these actuators, since it has a heavy weight of 110 kg and performs agile locomotion.

For any of these actuators, there should be an energy source to empower either the actuator itself or energy conversion facilities. Again, following categories are often used in mobile robots:

- **Batteries:** Commercially available with reasonable price, but usually with low energy storage factor (energy/weight).

- **Solar Panels:** They are a bit more expensive and geometrically space consuming, but introducing longer operation time. Since they can not generate large amounts of power with limited size, they are rarely used in legged robotics.

- **Gasoline:** Has a large energy storage factor comparably and often used for long time operation. It is not reasonable to use them in electrical robots as another converter is needed to convert chemical to electrical energy which is heavy. However in pneumatic or hydraulic robots, they are reasonable as compressors could run with gasoline engines. The efficiency is usually low and thus one needs to re-fuel the robot. It worth denoting that controlling a gasoline engine (especially position control) is difficult in joint level for legged robots.

Having these categories in mind, for a heavy legged robot being capable of doing dexterous tasks, hydraulic actuators are better in terms of power. For energy source, gasoline compressors are newly becoming common (used in BigDog as well), although some people use batteries for hydraulic robots nowadays.

### 1.1.3 Compliance

From the view-point of control, one is interested in making the robot interact with the surrounding environment friendly. This physical interaction is important in case of hazards for human operators in industrial robotics. In wheeled and aerial robots, the interaction does not produce very hard impacts on the robots continuously. However in legged robots, each step makes an impact on the whole body. These shocks can be harmful for the environment and for the robot itself. Beside mechanical techniques used to alleviate these effects such as using soft contacts on foot, people use more sophisticated hardware and low-level control methods:
• **Mechanical Spring Dampers:** These elements are attached to the main joints in the robot to absorb shocks in joint level and to decrease their stiffness mechanically. However, these elements make the control problem more complicated, since they introduce some dynamics to each joint.

• **Joint-level PD feedbacks:** This control architecture makes a joint following a desired trajectory while absorbing shocks actively. It depends on the gains of PD feedback of course and there is a trade off between preciseness and compliance.

The first category is usually called **passive compliance** while the second one is called **active compliance.** From the viewpoint of control it is better to avoid passive elements as they degrade tracking properties. However these elements are widely used where preciseness is not of crucial importance. For example in quadruped robots, since the robot is statically stable, one might not need exact trajectory following. On the other hand in biped locomotion, better tracking helps stability of the robot. Note that one might use both of these approaches together and include passive dynamics in calculations and compensate for that while generating actuator profiles. For a good comparison of passive and active compliance, see [4] in the area of industrial robotics.

### 1.1.4 Agility

Agility is a requirement that imposes various limitations in the design of a robot, both mechanically or in software. Either in an industrial task like an assembly line where production speed is important, or in a mobile robot assigned to collect information for example in the battle-field, an engineer should have a clear answer to the following problems:

- Are actuators able to produce high powers?
- Do the power supply provide high instantaneous power?
- Does the mechanical structure tolerate shocks and fast movements?
- How harmful are expected/unexpected shocks to the actuators?
- How fast the control algorithm is?

In addition to actuator types, maximum torques and control bandwidth, an important question in designing agile robots is shocks produced in each step. On control part of the robot, the aim is to design more compliant and precise algorithms, since higher speeds need better prediction and also more compliance so that not to
be harmed in long term operation. For each joint depending on the type of actuator and the very low-level and fast control loops, we might see either a position/velocity or force controlled actuator. The former requires a position/velocity value per time step while the later needs a torque value. One can also combine them together in case of a torque controlled robot as:

\[ \tau = \tau_{desired} + K_p(q_{desired} - q) + K_d(q_{desired} - \dot{q}) + K_i \int (q_{desired} - q) \]  

(1.2)

This requires sensor readings, but controls the compliance of the robot. Again feedback gains play an important role. Note that one might omit the \( \tau_{desired} \) from the equation and probably other elements like \( K_d \) or \( K_i \) to make the joint position controlled. Often a nonzero \( \tau_{desired} \) means that the control algorithm uses the exact rigid-body model of the structure to produce feed-forward torques which compensate interconnections between different links of the robot. We will discuss this later in chapter 2. Using the exact model of the robot may help predicting its behavior and to plan rapidly while solely counting on feedbacks has the disadvantage of settling time or tracking dynamics. The model also helps us produce required compensating torques and avoid large feedback gains to have the same tracking precision. So for agile robots and especially for legged ones who have more sophisticated structures, it is beneficial to have the model of the robot assuming enough calculation power. We will discuss this in chapter 2 as well.

### 1.1.5 Robustness

This topic has different meanings in each category of robots. While in aerial or wheeled robots, robustness means following a specific trajectory and being able to recover from perturbations, in the area of legged robots it has a wider meaning. For aerial robots, they should not fall and be robust in different weather conditions. The control problem is therefore a continuous problem by nature. In wheeled robots the case is similar except that they should not accelerate as fast as they lose wheel contacts. Again the problem has continuous states. In legged robotics however, although the same problem appears like wheeled robots, but we have to deal with hybrid states regarding different arrangement of feet on the ground at each instant of time. The case becomes like wheeled robots only if the robot stands and does not walk. Based on these arguments, the division would be standing (not-hybrid) and walking (hybrid) robots where in later, two different types of walking control are proposed for legged robots: static and dynamic walking.

By definition, static walking means that the center of gravity (CoG) always falls within a support region of feet. Either in single support, double support or more legs on the ground in quadruped or hexapod robots, we can identify a polygon
formed be ground contacting legs and we can control the robot using redundancies to move the CoG within this polygon. In Biped locomotion which is the point of interest in this project, static control will be therefore a series of continuous and hybrid actions to create new regions of support and to move CoG forward within these regions. Usually, this approach requires large feet and strong ankle joints as in [5], the robot can not reach high speeds and consumes high power.

Dynamic walking in contrast, means that CoG can sometimes fall outside support polygon and the robot is able to maintain balance by taking additional steps to make a new polygon if required. This concept was first introduce by McGeer [1] who proposed the passive walker of Fig.1.1 a). The robot is gravity powered in the simplest case and has feet that resemble wheels. Using two feet rigidly connected together, they reduce the dimensionality of the problem to 2D and analyze its stability by linearization of step-to-step state transition. Obviously, there is no control over CoG to fall inside a polygon. The idea is simple for 2D locomotion, but for 3D bipedal robots is not. We will refer to this type of walking later in chapter 2.

1.2 Contributions of this work

In this project, we would like to propose a new method for biped walking control. The exact platform of simulations will be later introduced in chapter 3. However, the goals are:

- **3D walking.** Many efforts have been made to control gaits and making them stable in 2D. While walking gaits are important in coronal plane (to the front), it is important to maintain balance in sagittal plane (not to fall to left or right).

- **Compliance.** As described before, in the area of legged robotics, per step we have a shock that should be absorbed in the system to avoid self-harming. Although mechanical tricks ameliorate the situation a bit, our control method should be compliant as well with previously mentioned definitions.

- **Fast walking.** The maximum walking speed depends on many parameters such as leg length, actuator power and type of terrain. We assume enough processing power and also friction in the environment and would like to achieve 0.5m/s which is reasonable for human-sized bipedal robot under control in this project.

- **Push recovery.** Our method should be able to reject pushes both in stance mode (not hybrid) and walking mode (hybrid).
• **General method:** We would like to propose a method that requires minimal off-line optimization and tuning of parameters. The trade off though is on-line calculations which should not be heavy. Using the model in the system will bring the knowledge people obtain by learning or off-line optimization, but it has the disadvantage of heavier mathematical calculations.

In chapter 2 we will review literature regarding various methods proposed for biped locomotion and compare them regarding above-mentioned goals we have. The simulation platform, humanoid robot properties and different packages used in our controller are later described in chapter 3. We will also introduce the odometry method we use for determining the position of the robot in general coordinates in chapter 3. After that, three different controllers are designed and tested which will be explained one by one. In chapter 4 we will describe the low level method used to produced joint torques, given desired Cartesian accelerations for the robot CoM and feet. Various tests are performed in this chapter to show properties of this controller. We will also propose a *static walking* algorithm and see how slow and un-natural it is. In chapter 5 we try to improve the controller by taking future of the motion into account and plan in advance. This will make the walking look more natural, but still the robot is slow and fragile when exposed to external pushes. Finally in chapter 6 a better controller will be introduced which shows full *dynamic walking* and satisfies all constraints and goals. It can produce natural walking, recover strong pushes and achieve high speeds. A qualitative comparison between different controllers is performed in chapter 7 to wrap up the project. We also list possible future improvements to the algorithm in chapter 7.
Chapter 2

Literature review

In this chapter, we would have a brief review on different approaches proposed in literature to control biped walking. We can categorize legged robots as:

- **Monopod**: These robots namely have one leg and hop on the ground dynamically. Although there exist researches considering a foot for them like the pneumactic-driven robot in [6], most of researches assume point contact with the ground. Among various projects, the 3D one-leg hopping machine of Raibert [7] back in 1984 is outstanding as it can stabilize using very simple laws, equipped with very basic computation facilities. After that, most of researches focused more on Spring Loaded Inverted Pendulum (SLIP) model like [8] and [9] to investigate stability properties of hopping. The SLIP model together with Linear Inverted Pendulum Model (LIPM) used for walking has helped people replicating CoM trajectories and ground reaction force (GRF) profiles. Studying these models for single leg is important as biped or quadruped robots are composed of single legs. One can also simplify these robots to SLIP or LIPM in specific gaits. Many people use symmetric hoppers (two synchronized legs) or connect them to a rotating boom to solve the problem in 2D which is not desired in our case.

- **Biped**: These robots namely have two legs and resemble humans and some animals like monkey, ostrich and congreve. Unlike others, congreve uses two synchronized legs which resemble the case of monopod with large foot. For the rest, stance, walking and running are popular gaits that humanoid robots want to imitate. In the rest of this chapter we will introduce different approaches to control bipedal robots. These approaches might be extended to more legged robots as well.

- **Quadruped**: Namely having four legs which resembles many animals. Regarding the number of legs, one might think of many different time sequences
like throating and binding in horses for example. These rhythms are mainly hybrid states while questions about control and compliance refer to continuous states. Among various quadruped robot, BigDog [3] is the most famous one nowadays introduced in chapter 1.

- **Hexapod, Octapod etc.**: These robots are more stable compared to previous ones as we discussed in chapter 1. The approach of keeping CoG inside support polygon is easier to apply on these kinds of robots. RiSE [10] and RHex [11] are two good samples being able to climb and do dexterous locomotion respectively.

Having this classification in mind, it is obvious that having more legs makes the robot more stable and thus, less complicated controllers are required. Researches like [1] and [2] are more focused on minimizing energy; on the other hand, maintaining stability is yet a big problem in bipedal locomotion if required to perform agile and dexterous tasks like running on complex terrain. In the rest of this chapter, we will briefly categorize various methods used in the control of legged robots while emphasizing on aspects being important in biped robots.

## 2.1 Model-Free Methods

In these approaches, we assume that there is no model of the robot, neither kinematic nor dynamic. Although people might include some learning process for evolution of an optimized controller like [12], the intrinsic simplicity of these methods make them a proper choice for more stable robots from quadruped and so on. Among these methods we can enumerate:

### 2.1.1 Central Pattern Generators (CPG)

These controllers are based on generating proper signals for different joints centrally, assuming that these joints have proper tracking properties. Sensor information from environment could be added to the loop, but difficult to deal with systematically. The CPG is based on a set of dynamical systems which produce different signals with variable amplitude, phase and frequency, inducing different gaits in the robot. Fig.2.1 shows a sample architecture used in Salmander robot in [13]. The role of interconnected network is to generate properly phased signals for different joints.

As discussed before, quadruped robots are categorized among stable robots regarding polygon of support, promising for simple control approaches such as CPG. Nevertheless, people have applied CPGs on biped robots as well in a limited way as in [14]. This work uses Neural Oscillator Model with reinforcement learning to obtain parameters online. A more recent work [15] also uses a modifiable
controller framework based on neural oscillators and optimizes various parameters. To compare with our own problem, both of these works have implemented their algorithms limitedly on big-foot and small-sized humanoid robots which inherently have good stability properties. Moreover, the specific tuning of parameter is either obtained off-line which will work for a limited range of movements or obtained online via a learning process which is time consuming and cannot recover from sudden changes. It is worth mentioning that these approaches include sensor position readings in their CPGs to improve tracking properties which might degrade the compliance of the robot. The case becomes worse if the robot is position controlled.

2.1.2 Reflex based control

The idea here is to use local reflexes in different joints [16]. By proper tuning of these parameters, a walking sequence emerges automatically without enforcing any specific trajectory to be followed. The basis is SLIP model to generate walking behavior which is shown in Fig.2.2.

To control the robot and generate proper actuator torques, the SLIP model is evolved using a couple of muscles, inspired from human. Each muscle $m$ is stimulated by $S_m(t)$ as:

$$S_m(t) = S_{0,m} + G_m F_m(t - \Delta t_m)$$

(2.1)

Where $S_{0,m}$ is a pre-stimulation and $F_m$ is muscle’s force (coming from sensor) which is time delayed and gained by $G_m$. The muscle therefore produces an output force given by the following differential equation [17]:

$$\tau dACT(t)/dt = S_m(t) - ACT(t)$$

(2.2)
Where $ACT(t)$ is the output force being projected to the joint regarding geometry of muscle connection. This formulation for muscle is called $F^+$, i.e. positive feedback and is basis of reflex properties shown in Fig.2.2 B. Negative feedbacks could also be introduced by adding terms to the stimulation function. For example in Fig.2.2 C, the VAS muscle gets influenced negatively if the knee joint extends beyond $170^\circ$. One can also penalize stimulation of some muscle by adding measures of the length of another muscle to improve stability. These interconnections between muscles are defined inspiring from human and parameters are either tuned by hand or optimized off-line. Swing behavior and orientation control of the torso could also be realized by influencing some muscles with these variables. As an example, swing could be done by disabling some positive reflexes in some muscles and make the knee bend for some moment. We could then switch to stance mode by enabling it again. The total walking behavior comes out of all these reflexes and the only parameter we enforce to the system is our mode switching law.

The system becomes stable in a very simple manner and works even with a rough initialization of parameters. However, it only works for a limited range of gaits and might need re-tuning of parameters. The compliance also highly depends on feedback and feed-forward gains which are directly related to stability. Although the model looks promising, defining those relations between muscles is somehow intuitive and is not done systematically.

2.2 Kinematic-model based methods

Beside model free methods which highly depend on tuning/optimization, one could benefit from very simple kinematic model of the robot to improve performance and
avoid tuning vast of parameters in different parts of the system. The kinematic model of a rigid body means geometrical relations between different points of the system in Cartesian space and degrees of freedom in the system. A rigid-body consists of a set of links and joints which might be actuated or not. In an industrial context since usually the robot is fixed based, the number of degrees of freedom in the system depends on the number of joints. However in a mobile robot, we have 3 more degrees of freedom in 2D and 6 more in 3D case regarding position and orientation of the robot in general coordinates.

The topic of kinematic problem is to relate joint angles with Cartesian position of the end effector or any other point on the links of a serial-chain. The position and orientation of a point could be related to robot’s degrees of freedom by:

\[
x_c = f(x, q)
\] (2.3)

Where \( x_c \in \mathbb{R}^6 \) are position and orientation of an endpoint, \( x \in \mathbb{R}^6 \) are degrees of freedom of the robot in Cartesian space and \( q \in \mathbb{R}^n \) are \( n \) joint variables of the robot. This is the forward kinematic equation depending on the geometry of the robot. Based on kinematic data, three main approaches are seen in literature to control an articulated body.

2.2.1 Inverse kinematics

In this approach, a desired trajectory is defined in Cartesian space for foot and pelvis of the robot and inverse kinematics gives desired joint angles and motions realizing that motion. Trajectories are usually described by splines and optimized regarding some objective function like energy. This method is sometimes augmented with rules to stabilize the robot as in [18] or to control the zero-moment-point falling inside support polygon like [19]. The main subject of inverse kinematics is to solve eqn.2.3 reversely to find desired joint motions. Sometimes it is easy to find a closed form solution analytically which requires the system to have unique solution. Libraries like IKfast \(^1\) help to generate the solutions in a fast manner while there exist other numerical method which are based on integration. Note that by derivating the eqn.2.3, one can relate Cartesian speeds to joint speeds by:

\[
\dot{x}_c = \dot{f}(x, q) = J(x, q) \dot{x}\] (2.4)

Where \( J(q) \) is called Jacobian. Some approaches start from a known solution in joint space and try to navigate the end-effector on a desired path by speed control.

---

\(^1\)http://openrave.org/docs/latest_stable/openravepy/ikfast/
Inverse kinematics approaches heavily depend on trajectories and are usually used in position-controlled robots. Stabilizing methods like ZMP which we discuss later are used to control the robot and can not produce fast motions and compliance.

2.2.2 Zero Moment Point (ZMP)

Remember the discussion on the stability requirements in biped robots in previous chapter. One approach to make sure that the robot is always stable is to keep the center of mass (CoM) inside a support polygon formed by feet all the time. Ankle torques provide controllability inside feet region. Such method will however lead to a very artificial walking. We will test such algorithm in chapter 4 and see that it is too slow and with high sagittal motion. To avoid these drawbacks, the concept of ZMP was introduced in robotics whose contribution was introducing gaits which are dynamically balanced. If not balanced, the robot collapses around the foot edge and looses full contact. This might happen a lot in human motion, when walking fast, but ZMP based gaits can not control such motion. In case of dynamic balance by definition, ZMP is the center of pressure CoP of the floor reaction force given by:

\[
\text{CoP} := \frac{\sum_{i=1}^{N} p_i f_{iz}}{\sum_{i=1}^{N} f_{iz}} \tag{2.5}
\]

Where \( p_i \) is the position of a point and \( f_{iz} \) is the vertical component of ground reaction force at that point. One can also integrate these forces continuously inside a convex support region which implies that CoP always falls inside as shown in Fig.2.3.

![Figure 2.3: Definition of ZMP (taken from [20])](image)

In double support phase (both feet on the ground) the same point can be calculated, though the vertical force is zero in some region. Provided that we have a convex region, the CoP always falls inside. If we calculate the torque around ZMP, it is easy to show the \( \tau_x \) and \( \tau_y \) components of the torque are zero and that is why this point is called Zero Moment Point.

We have another way to calculate ZMP, using linear momentum \( \mathcal{P} \) and \( \mathcal{L} \):

\[
\begin{align*}
\mathbf{f} &= \dot{\mathcal{P}} - Mg \\
\mathbf{\tau} &= \dot{\mathcal{L}} - \mathbf{c} \times Mg
\end{align*} \tag{2.6}
\]
Where $\mathbf{f}$ and $\tau$ are external forces and torques applied to the system, $\mathbf{c}$ is center of mass, $M$ is total mass and $\mathbf{g}$ is gravity vector. Supposing that external forces are acting at ZMP point at $\mathbf{p}$ we have:

$$\tau = \mathbf{p} \times \mathbf{f} + \tau_{ZMP} \tag{2.7}$$

From this equation, one can calculate $\mathbf{p}$ such that the first two components of $\tau_{ZMP}$, i.e. $\tau_{ZMP,x}$ and $\tau_{ZMP,y}$ become zero. Note that the ZMP calculated in this way might fall outside the support polygon which implies that the assumption of full foot contact had been wrong. So one should recalculate ZMP and will find that ZMP has been located on the edge of support polygon, making the foot rotated. Note that a stationary robot will have CoM projected exactly on ZMP. However a walking robot might maintain its ZMP inside polygon while having CoM outside.

Modern ZMP based control approaches usually try to realize prescribed ZMP trajectories by creating proper body motion from inverse kinematics mentioned in previous subsection. Formerly, people were trying to rather realize prescribed leg trajectories and control ZMP beside by moving a big mass which was not making a natural walk. In both methods, important assumptions are:

1. The robot should be fully actuated and position controlled.
2. Extra constraints should be introduced to solve the redundancy problem if $q$ is larger than 3. For example controlling posture and height of pelvis.

Therefore, we can conclude that although having ZMP in mind definitely helps improving the walking stability as foot should not roll and create edge contact, but ZMP itself can not help performing dexterous and agile tasks as required. We also want the robot to be complaint which is not easy in case of position controlled robots. For a complete review of ZMP and CoP, please refer to [21] and [22].

2.2.3 Potential fields

The idea of using potential fields was first introduced by Khatib in 1987 [23]. In the context of industrial robots, one is interested in navigating the end effector of the robot in the surrounding environment full of other robots or obstacles. The goal point has attracting properties while other obstacles are repulsive. Therefore, the gradient of a total potential function determines the direction that could be followed by the end effector. This approach is called Operational Space Control as it controls the robot in the workspace, not in joint space. An example of operational space obstacle avoidance is shown in Fig.2.4.

Recall from previous section that one can relate Cartesian velocity of end effector to joint velocities by eqn.2.4 as:

$$\dot{\mathbf{q}} = J^{-1}(\mathbf{q}) \dot{\mathbf{x}}_e \tag{2.8}$$
In case of fixed-based robotic manipulators, $\dot{x}_c$ is the desired motion determined by operational space control. There are three main problems appearing:

1. **Redundancy**: The problem appears when total number of degrees of freedom are more than 6 in 3D case. In this situation, $J^{-1}(q)$ is not defined and actually has infinite solutions. One might use $J^+(q)$ where $+$ is a generalized pseudo-inversion method to find an optimal solution. Using weighting matrices, one can also emphasize the role of some joints. A good analysis of stability is proposed in [23] as well.

2. **Singularities**: In some postures $J(q)$ might not have full rank, meaning that there is no delta-motion in joints that can produce a desired motion at the end-effector. These singularities usually happen at the borders of robot’s workspace and people avoid them by shrinking workspace. However, this problem needs a constrained control which is not possible with pseudo-inversion methods. It is also possible to use potential fields for avoiding joint limits and singularities and they can produce a proper weighting matrix. Similar concept could be found in [24] and [25].

3. **Local minimum**: There could be some points where the potential function return zero gradient and thus no motion. Although adding random movements might help in some situations, but more complicated navigation methods are needed.

To conclude, this method works in simple environments very well and is more applicable on industrial robots. In legged robots however, we have 6 DoF regarding mobility which are not actuated. Although for humanoid robots, a foot at the end of leg adds 6 constraints, but the robot is not controllable in a large workspace since the foot has finite size and we have ZMP constrains introduced before. In case of point contacts, each add 3 translational constraints and the robot becomes
controllable with minimum of three contacts to make a region of support. So the problem in mobile robots is actually navigating the main body through null-space of contact Jacobian which differs with industrial robots. The application of potential fields on legged robots is very limited. \cite{26} and \cite{24} use these fields to navigate a legged robot while \cite{27} uses them to find a stable walking gait. Potential fields are therefore mostly applicable to manipulators, and if it appears in legged robotics, it is used in planner level, not low level control. However operational space control is a more general concept that could be used in legged robotics as well, which will be described later.

\subsection*{2.2.4 Virtual Model Control (VMC)}

This approach uses virtual components to compensate existing redundancy in the robot. In typical legged or humanoid robots, there are several actuators used for different legs. VMC method puts virtual elements between contact points. These elements which are spring-damper pairs, generate forces $F$ that are translated to actuator torques $T$ using contact Jacobian $J_C$ by following equation to produce a desired motion based on contact behaviors required.

$$\mathbf{T} = J_C^T \mathbf{F}$$

By modulating virtual elements then, a robot can move and work in different hybrid states. More specifically, new variables appeared in equations are set-points of springs. Here, we will briefly explain two instances of works profiting from this method.

In this work \cite{28}, the concept of virtual components is used in two ways: stance control and motion. Stance control is done with the aim of granny walker mechanism which is shown on Fig 2.5

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{spring_turkey_robot.png}
\caption{(taken from \cite{28}) Spring Turkey robot: Left: Granny walker mechanism used in stance control, Right: Dogtrack Bunny mechanism used for locomotion}
\end{figure}

Two virtual components help to control attitude and height of the robot using redundancies. For motion control, a virtual component is used between robot and a constantly moving object. This so called “Dogtrack Bunny mechanism” is shown
They combine these two approaches using a state machine which is shown in Fig. 2.6. The benefit of using this method is the intrinsic flexibility in stance control and prioritization of tasks in low level controllers. However, exact position control of CoG is complicated and this method will guaranty position and attitude of the base to vary in a small range.

The second piece of work is [29], where a control model is proposed and applied on a real robot based on virtual components. At first, details are explained for a 2D biped robot and then, it is extended to a quadruped robot called AloF, developed at ETHZ [2].

Figure 2.6: (taken from [28]) Spring Turkey robot: State machine for using different control methods

Figure 2.7: Virtual model control on ALoF (taken from [29]), Left: 2D model shown in stance phase with specific configuration of a virtual component for this phase, Right: the actual robot and all virtual forces created by virtual components between contact points

2Swiss Federal Institute of Technology Zurich
Features of this work could be summarized as follows:

- There is no sensor for contact forces and they use contact dependent virtual components.
- A higher level controller dictates positions of CoG and lower level controllers follow these commands.
- Three main tasks are described in this work: stance control, motion control and standing up.

In Fig. 2.7 on left, we can see a virtual component defined between base and mid-point of contacts which is used in stance phase, when both legs are on the ground. On right image, necessary foot navigating forces to be generated are shown between feet.

Overall, having reviewed some instances that use VMC, we come to the conclusion that this method has many advantages in particular simplicity and less computations needed. However, it is blind to the dynamics of the system and may not track desired trajectories as we want. Moreover, contact forces are not guaranteed to be optimal and thus the robot’s interaction with environment is not possibly as compliant as what we are looking for.

### 2.3 Dynamic-model based methods

In previous sections, we were mostly discussing methods based on a very simple model of the robot. The matching between the model and robot was done using heavy optimizations, hand tunings or online learning processes. Although these methods might be able to control the robot for a specific task or in a restricted region of states, they may fail in cases where working conditions are varied. In fact, there is a trade off between these concepts. Better tracking and thus controllability could be achieved by increasing feedback gains, however the robot becomes stiff and interacts stiffly with the environment. Also one should expect enough time for simple feedback controllers to reach desired set points. Effects like gravity and Coriolis forces are not systematically compensated with feedback loops. These limitations therefore motivate engineers to benefit from dynamical model of the system which include all the information about a rigid-body behavior and effects of other external forces like gravity and constraints. This information helps predicting motion of the robot in future and calculating exact required torque to achieve a desired speed or position. Knowing this torque, one can reduce feedback gains and make them only responsible for perturbation rejection. It implies that these methods are mostly used on force/torque controlled robots. In this section, we
will introduce the formulation and various approaches taken in legged robotics to benefit from exact model of the robot in the control loop.

### 2.3.1 Inverse dynamics

Full dynamics of a robot can be expressed in an equation called Equation of Motion (EoM):

\[ M(q) \ddot{q} + h(q, \dot{q}) = \tau \]  

(2.10)

Where \( q \) is system variables, \( M \) is the mass matrix, \( h \) is gravitational and Coriolis forces and \( \tau \) is actuator torques. \( q \) consists of all DoF in joints and also 6 Cartesian DoF regarding the position and posture of a floating based robot. To simulate a system, an ODE or numerical differential equation solver needs to know accelerations or derivatives of states. So forward dynamics refers to giving \( \tau \) to the EoM and obtaining \( \ddot{q} \) to integrate the system and obtain new states knowing initial conditions. In contrast, inverse dynamics refers to giving desired \( \ddot{q} \) to EoM and obtaining \( \tau \) to be applied to the system which realizes that acceleration. The \( \ddot{q} \) is coming from known trajectories here.

### 2.3.2 Projected inverse dynamics

The idea of projected inverse dynamics was first formulated in [30]. The Eqn.2.10 is written for a floating rigid body which does not have any contact with environment. Although this formulation might be useful in limited cases in space robotics, it cannot be applied to legged robotics where they might have multiple contact with the environment at each instance of time. A more complete version of Eqn.2.10 is:

\[ M(q) \ddot{q} + h(q, \dot{q}) = \tau + J_C^T(q) \lambda \]

(2.11)

Where \( J_C \) is the Jacobian of contacts and \( \lambda \) is contact forces. The assumption of this formulation is therefore point-wise contact which is not realistically true, but works for many legged robots unless they do not have foot. We will discuss later how we deal with this issue as our simulated robot has foot. In the new formulation, one has to know where the contact point is and also what is the exact value of \( \lambda \). A major problem for controlling the robot is the fact that contact forces depend on actuator torques and vice versa. Since physical sensors are not so accurate, suffer from considerable noise and also have a delay, control algorithms tend not to rely on them. A common approach is to calculate contact forces by knowing actuator forces in previous step and use this estimation at current step to estimate contact forces and obtain new actuator torques. This method leads to unstable control loop in adverse situations, but is acceptable in fast enough
control loops, since joint angle sensors are reliable for calculating other terms in EoM. Note that in many legged robots, force sensors are at least used to determine if a leg is on the ground or not, i.e. determining hybrid states like \cite{29}.

This paper \cite{31} has proposed a new method. By Q-R factorization of Jacobian matrix \((J^T_C)\) at each time step and projecting the main equation of motion by \(Q\), we will have two different equations where there is no contact force in one of them. Using this equation, we can find \(\tau\) and control the robot satisfying contact constraints.

Another problem in controlling a robot is the fact that some joints might be passive and not actuated. In this case, some elements of \(\tau\) must be forced to zero. The most complete form of the EoM becomes:

\[
M(q)\ddot{q} + h(q, \dot{q}) = S^T\tau + J^T_C(q)\lambda
\]  

(2.12)

Assuming \(n\) to be number of DoF in joints, \(m\) to be number of passive joint which is normally zero and \(k\) to be number of constraint equations, variables are defined as:

- \(q \in \mathbb{R}^{n+6}\): all degrees of freedom
- \(M(q) \in \mathbb{R}^{(n+6)\times(n+6)}\): the floating base inertia matrix
- \(h(q, \dot{q}) \in \mathbb{R}^{n+6}\): the floating base centripetal, Coriolis and gravity forces
- \(S = [0_{n \times (m+6)} \ I_{n \times (n-m)}]\): the actuated joint selection matrix
- \(\tau \in \mathbb{R}^n\): the vector of actuated joint torques
- \(J_C \in \mathbb{R}^{k \times (n+6)}\): the Jacobian of \(k\) linearly independent constraints
- \(\lambda \in \mathbb{R}^k\): the vector of \(k\) linearly independent constraint forces if there exists any

Principally, the algorithm calculates QR decomposition of \(J^T_C\) first.

\[
J^T_C = Q \begin{bmatrix} R \\ 0 \end{bmatrix}
\]  

(2.13)

Where \(Q\) is orthonormal \((QQ^T = Q^TQ = I)\) and \(R\) is an upper triangular matrix of rank \(k\). Then it projects Eqn\[2.12\] onto null-space of constraints by multiplying with \(Q^T\). The result is decomposition of the rigid-body dynamics into two independent equations:

\[
\begin{align*}
S_cQ^T(M\ddot{q} + h) &= S_cQ^TS^T\tau + R\lambda \\
S_uQ^T(M\ddot{q} + h) &= S_uQ^TS^T\tau
\end{align*}
\]  

(2.14)
Where $S_c$ and $S_u$ are used to select the top and lower portions of the full equation.

$$S_c = [I_{k \times k}, 0_{k \times (n+6-k)}]$$

$$S_u = [0_{(n+6-k) \times k}, I_{(n+6-k) \times (n+6-k)}]$$

The main point is that contact forces are not appearing in the second equation. This allows to calculate actuator torques without knowing contact forces. The control law would be:

$$\tau = (S_u Q^T S^T)^+ S_u Q^T [M \ddot{q} + h]$$

Where $+$ denotes Moore-Penrose pseudo-inverse method as:

$$A^+ = A^T (A A^T)^{-1}$$

Which minimizes the norm of torque vector $\tau$. Questions to be answered in this method are:

- **Redundancy and Over-actuation**: The ideal case is to have $k = 6$ and no singularities in $J_C$. In case of $k > 6$ the robot is called over-actuated and thus, we have infinite number of solutions to apply. The role of pseudo-inverse is to find proper solution as here, Moore-Penrose minimizes the total effort. One could also think of minimizing constraint forces or redistribute forces between actuators by a weighting matrix to put more emphasis on some actuators if required [32]. In case of $k < 6$ the robot is called under-actuated like the situation where a robot is in ballistic motion without any contact and $k = 0$. In this case, one should take care of specially Cartesian accelerations, since the formulation of Eqn.2.12 is written in a fixed-frame attached to the world and not in center of mass coordinates frame. Thus, a small acceleration (for main link-body) not consistent with ballistic motions might cause legs for example to extend rapidly.

- **Calculation budget**: Note that with simple formulation of Eqn.2.10, there is no need for matrix inversion in case of inverse dynamics. However the amount of calculations required to calculate the pseudo-inversion here is important. Basically there is one QR factorization and one inversion which is done with another QR factorization more efficiently. For the first QR factorization corresponding to $J_C^T$, the matrix is not full square and the Householder method works better [33]. This method is known to have better numerical stability. For the second one since the matrix is square, we can use the well known GramSchmidt projections [34] which are computationally lighter. Overall, the worst is householder which takes $O(n^2)$ in terms of floating-point multiplications, being reasonable in simple robots and not so fast control loops.
Based on the following discussions, one can conclude that:

- The method is fast.
- No need to know constraint forces.
- Intrinsic optimization.
- Gets joint accelerations as input.
- Not possible to add joint angle or torque constraints. Although people handle them by saturators, but this is not a systematic way.

The major disadvantage therefore is that one should generate desired accelerations for joints which is difficult regarding the fact that they should be constraint consistent:

\[
\begin{align*}
x_c &= f(q) = \text{const} \\
\dot{x}_c &= J_c \dot{q} = 0 \\
\ddot{x}_c &= J_c \ddot{q} + \dot{J}_c \dot{q} = 0
\end{align*}
\]

One can combine Eqn.2.18 with Eqn.2.12 to omit \( \ddot{q} \) and obtain an operational space controller as Mistry formulated in [35]. He extended the traditional formulation of Khatib for manipulators to floating-based legged robots which intrinsically have constrained nature. The exact formulation will be shown later, however the framework is not yet able to deal with actuator limitations or ZMP constraints systematically.

### 2.3.3 Trajectory optimization

In this approach, one considers the timespan between two hybrid events (like touch down or lift off) and optimizes trajectories regarding an objective energy function. This function has a horizon of N time-steps in future and a cost regarding the last state which ensures periodicity or another specific criterion. The problem formulation is more general than typical model predictive control (MPC) problems [36], and could also be non-linear. MPC is only a specific version which optimizes states and control inputs for a finite horizon and with some linear constraints. The objective function at time-step \( i \) is:

\[
V(x_i, i) = \Phi(x_N) + \sum_{j=i}^{N-1} r(x_j, u_j, j)
\]
Where $x_i$ denotes system state and $u_i$ denotes control policy we want to use at time-step $i$. $\Phi(x_N)$ is the terminal cost and $r(x_i, u_i, i)$ is the cost function. Successive states in the system are related together by:

$$x_{i+1} = f(x_i) + b(x_i)u_i$$

(2.20)

Which is known as forward model. At each time step, the optimization generates best $u_i$ being applied to the system. A well known approach is called Differential Dynamic Programming (DDP) [37] which minimizes $V(x_i, i)$ by iteratively going back and forth. Let us define a Q function as:

$$Q(i) = r(x_i, u_i, i) + V(x_{i+1}, i + 1)$$

(2.21)

DDP considers a second order local model of $Q(i)$ (in terms of $x_i$ and $u_i$). Given the current state of the system $x_i$ and starting from an initial guess of $u_i$, one can improve the control policy $u_{i}^{new} = u_i + \delta u_i$ from $argmax_{\delta u_i} Q(x_i + \delta x_i, u_i + \delta u_i, i)$. Having calculated the new $u_{i}^{new}$, we iterate until $x_N$ and obtain a new Q function and its second order local model. At each time-step therefore, we have the optimal local trajectory $x_{i}^{opt}$, control policy $u_{i}^{opt}$, value function $V^{opt}$ and the Quadratic function $Q^{opt}$. We apply $u_{i}^{opt}$ to the system and it runs until the next time-step, which is the time to re-optimize.

Discussions on convergence rate, number of steps $N$, backward propagation and the quadratic function $Q$ could be found in [37]. In [38] for instance, this approach is used for biped locomotion taking model errors into consideration. Model predictive control is dependent on the model of the robot. If constant disturbances appear in the system, it might seriously affect the performance or stability. The authors in [38] use a kind of online learning to augment $Q$ function by adding a $\Delta F(x_i, u_i, i)$ term which comes out of Receptive Field-Weighted Regression (RFWR) learning block [39]. Application of this approach could found in [40] for example where local optimal and periodic trajectories are optimized by this method. Then they are used to build a library of trajectories to control walking of 3D torque controlled robot called Sarcos. Similar method is taken by [41] to find trajectories for uncertain environments. These approaches are computationally heavy considering the fact that we have the full model of the robot used per iteration. One would be interested in running predictive control at a more abstract level. We will use MPC control in next chapters however with similar formulation and linear constraints, but different solver.

### 2.3.4 Hybrid zero dynamics

Among various model based methods proposed for biped walking, Grizzle has developed a well theory-supported method for generating stable controllers that
help make the robot converging to an stable orbit optimized off-line. The method
tries to reduce dimension of the system to few variables that usually represent the
state of the system in task space. Recall the Eqn.2.12 we mentioned before:

\[
M(q)\ddot{q} + h(q, \dot{q}) = S^T \tau
\]  

(2.22)

One can define the state of the system \( x = [q \ \dot{q}]^T \) and derive its derivative from
the previous equation as:

\[
\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} M^{-1}(q)[S^T \tau - h(q, \dot{q})] \end{bmatrix} =: f(x) + g(x)u
\]  

(2.23)

Now let us define the output function \( y = C(q) \) which is a function merely
depending on the configuration variables. The derivative of \( y \) along solutions
satisfying Eqn.2.23 does not depend on inputs due to 2\textsuperscript{nd} order properties of the
system. If one calculates the second derivative of \( y \), it yields:

\[
\frac{d^2y}{dt^2} =: L_2 f C(q, \dot{q}) + L_g L_f C(q) u
\]  

(2.24)

Where the matrix \( L_g L_f C(q) \) is called decoupling matrix and only depends on
configuration variables. From this equation, one can calculate the control law \( u^* \) as:

\[
u^*(x) = -(L_g L_f C(q))^{-1} L_2^2 C(q, \dot{q}) = -(L_g L_f C(q))^{-1} L_2^2 C(x)
\]  

(2.25)

With a series of assumptions, the set \( Z := \{x | C(x) = 0, L_f C(x) = 0\} \) is defined
as zero dynamics manifold and \( \dot{z} = f_{\text{zero}}(z) \) is called zero dynamics. The control
law of Eqn.2.25 therefore renders \( Z \) invariant for swing dynamics under study, i.e.
for \( z \in Z, f_{\text{zero}}(z) := f(z) + g(z)u^*(z) \in Z \). This is a very simplified formulation of
the work in [42]. The main idea is that given a function \( C(q) \) this controller takes
a reduced dimension sub-manifold of state space and controls the system in such
a way that the system remains in that manifold and the function \( C(q) \) remains
un-changed. This is called zero dynamics of the system. To make it more complete
and robust, the authors of [42] include impact dynamics as well and make state
changes controllable. If we represent state changes happening during impact by
\( x^\dagger = \Delta(x^-) \), one would be interested to define the set \( Z \) such that it can control
the new state \( x^\dagger \). Such effect is better shown in Fig.2.8.

Note that the full state of the system in Fig.2.8 is composed of \( z \) \in \( Z \) and \( x \) as the
rest of states in the system. \( \phi^\dagger(x, z) \) is also called the flow of the continuous dynamics
of the system. \( \phi^\dagger \) is periodic with period of \( T > 0 \) if \( \phi^\dagger(\Delta(x^\dagger, z^\dagger)) = (x^\dagger, z^\dagger) \) as shown in Fig.2.8 Overall, the reset map \( \Delta \) will change system state after impact
and the zero dynamics controller will bring it to the previous initial value. Note that:
The output function $C(q)$ is defined based on the Poincaré return map $^3$ refer to [42] for more details.

The stable orbit which is found by off-line optimizations.

The work of [42] is extended and applied to a real robot called RABBIT in [43], this time introducing the concept of exponentially stable orbit using a local Lyapunov function. To compare with other approaches, the advantages of hybrid zero dynamics are:

- One can use this approach on foot-less robots like the five-link model used in [42] or the RABBIT robot used in experiments in [43].
- The control law makes system stable benefiting from null-space of task which could not be done by simple PID controllers.
- Due to inclusion of impact models $\Delta$ in the $Z$ manifold, one can make sure that the system does not heavily depend on the timing of hybrid states. Look for assumptions and arguments in [42].

However this method has quite important disadvantages as:

- The optimal orbit found off-line is a good solution only locally and can not be generalized. So one needs to re-optimize if different speeds required.
- So far, all the works using this method are planar robots (like RABBIT) whereas the robot we want to control in this project should work in 3D. It is therefore not so straightforward to define a 2-dimensional output function as in [42] and we have up to 6 task space variables to control.

---

$^3$This map or matrix is actually defined over one period of the system and indicates the change of states in the system after one period if the initial state is perturbed slightly.
- The assumptions in [42] are quite strong, especially regarding number of degrees of freedom and number of general coordinates which directly relate to under-actuation problem and are not general everywhere.

The concept of zero dynamics is similar to that introduced in projected inverse dynamics in the sense that in both, one is interested to use null space of constraints or task spaces trajectories to improve stability. However this method is locally stable and might not accomplish agile and flexible walking we are looking for.

2.3.5 Operational space control

The idea in this method is to emphasize on task space variables and tracking. Unlike previous one, i.e. hybrid zero dynamics, operational space control does not depend on pre-optimized trajectories in joint space. Rather it might use optimized trajectories in task space or motion captured data. A planner produces task accelerations and this block generates proper joint torques to realize that task. The formulation of operational space control was first introduced by Khatib in [23] and used for manipulators. We prefer to summarize this method based on [35] as it includes floating based, under-actuated and constrained systems as well. Recall the full rigid body dynamics equation Eqn.2.12 which we mention again:

\[
M(q)\ddot{q} + h(q, \dot{q}) = S^T\tau + J_C^T(q)\lambda \tag{2.26}
\]

Where:

\[
S = \begin{bmatrix} I_p & 0 \\ 0 & 0 \end{bmatrix} \tag{2.27}
\]

And \(p\) is the number of actuated variables. We also have an \(m\) dimensional task \(x \in \mathbb{R}^m\) and \(k\) dimensional linearly decoupled constraints \(\lambda \in \mathbb{R}^k\). We also define task Jacobian \(J\) as \(\dot{x} = J(q)\dot{q}\). Khatib’s formulation for full actuated \((n = p)\) and unconstrained systems \((k = 0)\) is:

\[
\Lambda \ddot{x} = \Lambda(JM^{-1}h - J\dot{q}) = F \tag{2.28}
\]

Where \(\Lambda = (JM^{-1}J^T)^{-1}\) and \(F\) is an external force applied at end-effector (i.e. task point). Eqn 2.28 is in fact the dynamics of the system in Cartesian space. To find a control law for redundant manipulators where \(m < n\), we have:

\[
\tau = J^TF + (I - J^TJ^T\#)\tau_0 \tag{2.29}
\]

Where \(F\) is obtained in Eqn 2.28 and \(J^T\#\) is in fact generalized inverse of \(J^T\) as:

\[
J^T\# = (JM^{-1}J^T)^{-1}JM^{-1} \tag{2.30}
\]
Note that $\tau_0$ does not affect task space dynamics and the motion resulted by $\tau_0$ is in fact decoupled. So overall, given a desired acceleration $\ddot{x}_{\text{des}}$ for task, one can obtain $F$ from Eqn. 2.28 and use Eqn. 2.29 to obtain proper torques $\tau$ to realize this acceleration without knowing anything from joint trajectories.

For constrained systems, we can use an orthogonal projection operator matrix $P$ so that to cancel out $J^T_C(q)$ from EoM. Therefore $P$ is selected such that $PJ^T_C(q) = 0$ and $P = P^2 = P^T$ and it could be computed by $P = I - J^T_CJ_C$ where + indicates Moore-Penrose pseudo-inverse. Now the EoM becomes:

$$PM\ddot{q} + Ph = P\tau$$  \hspace{1cm} (2.31)

Where we are interested in inverting $PM$ which is rank deficient because of $P$. However we have additional equations from constraints:

$$\begin{align*}
(I - P)\dot{q} &= 0 \hspace{1cm} (2.32) \\
(I - P)\ddot{q} &= C\dot{q}
\end{align*}$$

Where $C = (d/dt)P$. Since above equations are orthogonal to Eqn. 2.31 we can add the last one to Eqn. 2.31 obtaining:

$$M_c\ddot{q} + Ph - C\dot{q} = P\tau$$  \hspace{1cm} (2.33)

And $M_c = PM + I - P$. Aghili in [30] explains invert-ability and uniqueness of $M_c$ and finally like the unconstrained case, the same task space dynamics and control law yields.

Mistry has extended the formulation to under-actuated robots as well. If $\tau = S\tau$ where $S$ is defined in Eqn. 2.27 and $n - p > 0$ where $n$ denotes number of joint variables and $p$ denotes actuated ones, the robot is called under-actuated and it is not possible to control it with previous equations. Considering Eqn. 2.29 and $\tau = S\tau$, we have:

$$J^TF + N\tau_0 = SJ^TF + SN\tau_0$$  \hspace{1cm} (2.34)

Where $N =: (I - J^TJ^T)$. Solving for $\tau_0$ we have:

$$\tau_0 = -[(I - B)N]^+(I - B)J^TF$$  \hspace{1cm} (2.35)

Where again + denotes Moore-Penrose pseudo-inverse. The assumption therefore is having at least one valid solution for Eqn. 2.34 $\tau_0$ are in fact null space torques which now compensate for under-actuated joints. If the system had enough redundancy, Eqn. 2.34 has infinite solutions and we are minimizing $||\tau_0||$. One can also consider multi prioritized task and reformulate control laws which is discussed in [35] and is beyond the scope of interest in this brief review.

The advantage of this method is therefore not requiring to know trajectories. However we can not add constraints like CoP and joint torque limits in the body of optimization. We should deal with them separately. Also matrix calculations can sometimes become very heavy.
2.3.6 Whole body optimization

The previous control method was based on optimizing trajectories and control inputs to achieve better performance in future. There is another method which optimizes the current state of the robot and then one can use a predictive control in a more abstract way. The goal is therefore to reduce amount of calculations needed per time step. Specifically, one needs to determine center of mass translational accelerations $\ddot{C}_{\text{des}}$, angular momentum rate $\dot{\mathbf{L}}$, and foot translational and rotational accelerations $\ddot{P}_{\text{des}}$ and this optimization generates proper joint torques while minimizing accelerations and contact forces. This optimization therefore reduces the problem to determining Cartesian accelerations in task-space.

In [44], Stephens runs this optimization in two steps. First, since one can relate desired task-space accelerations directly with contact forces, he solves a Quadratic Problem (QP) to optimize for contact forces $\mathbf{F}$. Using the same notations as before, the dynamics of the system abstracted in CoM level could be written as:

$$
\begin{bmatrix}
    D_1 \\
    D_2
\end{bmatrix}
\mathbf{F} =
\begin{bmatrix}
    m \dddot{C}_{\text{des}} + \mathbf{F}_g \\
    \mathbf{L}
\end{bmatrix} \tag{2.36}
$$

Where $m$ is total mass, $\mathbf{F}_g$ is gravity force and:

$$
D_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} \tag{2.37}
$$

$$
D_2 = \begin{bmatrix} (\mathbf{P}_R)^\times & \mathbf{I} & (\mathbf{P}_L)^\times & \mathbf{I} \end{bmatrix} \tag{2.37}
$$

$$
\mathbf{F} = \begin{bmatrix} \mathbf{F}_R & \mathbf{M}_R & \mathbf{F}_L & \mathbf{M}_L \end{bmatrix}^T
$$

Where $\mathbf{P}_L$ and $\mathbf{P}_R$ represent positions of feet and $\times$ denotes left crossproduct matrix. Adding CoP and friction constraints which relate these forces together, Stephens solves a QP problem and determines $\mathbf{F}$ in double support phase while in single support, it is not needed since there are 6 equations and 6 unknowns. One can also solve a least square problem instead and release equality constraints if the problem did not have any valid solution.

For the second level of optimizations in [44], knowing contact forces $\mathbf{F}$, he solves the following unconstrained problem:

$$
\begin{bmatrix}
    \mathbf{M}(\mathbf{q}) & -\mathbf{S} \\
    \mathbf{J} & 0
\end{bmatrix}
\begin{bmatrix}
    \ddot{\mathbf{q}} \\
    \tau
\end{bmatrix} =
\begin{bmatrix}
    -\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{J}_C^T \mathbf{F} \\
    \ddot{\mathbf{P}}_{\text{des}} - \dot{\mathbf{J}} \dot{\mathbf{q}}
\end{bmatrix} \tag{2.38}
$$

Which is solvable with a pseudo-inversion method. Variables are defined similar to [2.12]. Using this method, Whitman [15] controls walking of a simple humanoid robot. His robot can recover from strong pushes and performs exact foot placement planning. However he has a large off-line optimization which produces lookup tables for such online performance. Discussing planners is not the subject of this
Another recent approach of this method is seen in [46]. They actually combine these two problems and solve them simultaneously in one QP problem. Given the same inputs $\ddot{C}_{\text{des}}$, $\dot{C}$, and $\ddot{P}_{\text{des}}$, they minimize only over $\ddot{q}$ and $F$ subject to contact acceleration constraints in 2.18, CoP and friction constraints, equality constraints in 2.36 and finally torque limitations:

$$\tau_{\text{min}} < \tau < \tau_{\text{max}}$$

(2.39)

Note that $\tau$ could be linearly calculated from 2.12. The advantage of this method is therefore simplicity of calculations. But one should use fast libraries to handle these amount of equations together. Overall, this method is useful because:

- It can handle all the constraints we were concerned about in previous methods.
- It considers dynamics of the system into account.
- We do not have explicit trajectories to be followed in joint level which increases compliance and simplicity a lot.
- Provides an abstract formulation for planners and thus, we can run predictive control in upper levels of controller more easily.

### 2.4 Conclusion

Having reviewed all model free, kinematic model and dynamic model based control methods, one can conclude that there is a trade off between using model and computations required. In early robots built at the time when processors were not powerful enough, people were using model-free methods to produce simple motions. Later as processors got stronger, kinematic approaches emerged since they required matrix operations. Nowadays, processors are strong enough to handle large amount of calculations and thus dynamic models are becoming more popular. Comparing different control methods, we can conclude that using dynamic-model based methods helps us considerably control a biped robot regarding the requirements of agile and robust walking.

In next chapter, we will briefly introduce the simulation platform and the robot we are using to demonstrate our method. We also use a couple of packages in our controller which will be introduced as well.
Chapter 3

Simulation platform

Apart from algorithms and methods used to control the robot, this chapter is dedicated to introducing the robot under control and other softwares used to simulate it. We also introduce different packages used in the controller and the perception method we use to determine the location of the robot. This project was first planned to have tests on a real robot, but due to some limitations we kept on working in simulations only. However, the simulation we are using is part of virtual robotic challenge (VRC) and is intended to simulate the real robot built by Boston Dynamics\[1\]. The robot is called Atlas and is simulated in the software as realistically as possible. Scenarios and tasks defined in VRC are also important and challenging which require integration of various perception, modeling, control and navigation algorithms. They include different complex tasks from simple walking to climbing ladder, walking on slopes and stairs and passing obstacles. The robot should also get inside a car and drive to a certain destination and manipulate different objects which include therefore very complex full body motions. In this project, we do not have perception and we aim at proposing a robust walking controller on flat ground or at most, with limited variations. Performing walking on more advanced terrains requires having a model from the environment which is beyond the scope of this project. In general, VRC is requiring teams to control a robot in a very realistic environment where various important factors are considered within simulation. So our controllers are expected to have robustness against various un-modeled effects not addressed in many similar simulation researches.

3.1 Atlas robot

This human-sized biped robot benefits from hydraulic actuators and torque control. Remember from chapter [2] that hydraulic actuators are usually used for high power

\[1\] http://www.bostondynamics.com/
and heavy robots. The Atlas robot is about 150kg which needs actuators far stronger than electric ones. The robot has 28 hydraulically actuated joints shown in Fig.3.1.

![Atlas Robot CAD Model](image)

Figure 3.1: The CAD model of Atlas robot simulated in this project. Joint names are shown as we need them in future. (Picture taken from Boston Dynamics.) Note that $x$ axis points to front while $y$ points to the left of robot.

Since the robot is torque controlled, it has torque sensors in all joints. It also has two IMU units on its pelvis and head, 6 DOF contact force sensors on each foot, two cameras and a laser scanner on its head. With these sensors, the robot has to build a map from the surrounding environment and navigate through as well as low level stance and walking control. These are actually requirements of the VRC competition. However we do not use cameras and laser scanners to perceive environment and instead, use our own odometry which will be explained later in this chapter. Note from Fig.3.1 that the robot has rather small sized feet compared to many ZMP controlled humanoid robots. Atlas is also much heavier than most of them which makes the control problem very difficult. However the robot has very strong actuators comparable with a real human and one needs to design a control method suitable for these properties. We will later in future chapters discuss calculation budget required by our control method. There has not been any documentation released on technical aspects of Atlas to verify feasibility. However we aim at designing a general method to be applicable on a wide range of robots.
3.2 Simulation software

For the purpose of Virtual Robotic Challenge (VRC), as many teams want to participate and prove that they have convincing control method, the organizers of this competition have developed all their models and simulation interface on a specific simulator. They also prohibit teams from accessing internal states of the software and rather, provide packet communication and predefined APIs to control the robot. They have also provided other computational resources which will be described later in this chapter.

3.2.1 Gazebo

The main simulator is ODE based, which ensembles many other simulators like Webots\(^2\). This simulator is called Gazebo\(^3\) and is initially developed at University of Southern California By Dr. Andrew Howard and his student. Gazebo reads object models from XML files and simulates them in an iterative manner. It supports many well known robots and there are various research groups using this simulator to validate their algorithms.

3.2.2 DRCSim

Based on Gazebo, organizers of VRC have developed lots of APIs, different environment models and the robot model as well to provide teams with a unique platform for the competition\(^4\). VRC has various navigation, locomotion and manipulation tasks. A simple example is shown in Fig.3.2.

Then in terms of locomotion, the robot should be able to do walking, foot placement, passing obstacles and going stairs and slopes up and down. We would like to achieve some of them in our project.

3.2.3 ROS

In order for teams to communicate with Gazebo, the well known communication library ROS\(^5\) is used and integrated with Gazebo to make DRCSim. This library is based on networked messaging and can be used in real robotic environments where there is a network of sensors, actuators and monitoring facilities as well as human operators. Although for the purpose of this challenge ROS may be used between human operators and robots (a sort of communication which is penalized to promote

\(^{2}\)http://www.cyberbotics.com/
\(^{3}\)http://gazebosim.org/
\(^{4}\)http://gazebosim.org/wiki/DRC
\(^{5}\)http://www.ros.org/wiki/
artificial intelligence), in low level control on the real robot, control algorithms are implemented on-board which is not delayed like ROS. In virtual challenge however, ROS is used in low level to prohibit teams from cheating. Different APIs developed to control the robot are readings of joint sensors, IMU and contact force sensors as well as images coming from cameras and laser scanners. Although some simple control modes are implemented by Boston Dynamics, but the main way to control the robot is restricted to commanding its joints. We will discuss this command in next chapter. Within the simulator, the robot is controlled with a loop of 1KHz and is thus able to execute commands coming in that rate. Of course one should consider actuator limitations as well. We will later comment on these timing of our control method. ROS has also other tools to visualize different packets and messages in the system and can even run a virtual model of the robot separately to visualize its sensor data like Fig. 3.3

3.2.4 CloudSim

In order to provide teams with good computational resources, the organizers of VRC have created a web application[^5] that launches simulation scenarios on a cloud system to facilitate parallel computation which is required for some control algorithms that are based on optimization as discussed in chapter 2. For the purpose of this project as we do not use (offline) optimization based control method and thus, we will not use these computational resources.

[^5]: http://gazebosim.org/wiki/CloudSim

---

34
3.3 Control packages

All previously mentioned softwares and libraries were used to simulate the robot. On controller’s side also, we use some libraries for modelling the robot and doing some fast on-line per time-step optimization. We will have a brief introduction here and discuss later how we formulate our problems. Other Algebraic libraries for vector and matrix manipulation are simple functions written by the author himself.

3.3.1 SD/FAST

This package is actually used to derive an exact and closed form model of the robot to do necessary computations for equation of motion (Eqn. 2.12). As we explain our method later, we will see that we only read the model from the library, neglecting various powerful properties in terms of fast calculations. In fact, SD/FAST is capable of simulating the model forward with an internal ode in a very fast manner which is used by some methods like trajectory optimization method described in chapter 2 among methods benefiting from the dynamics model of the system. SD/FAST can be used in analysis and design studies of different mechanical systems composed of rigid-body links and joints which are driven by forces and torques and have constraints. The library gets a very simple description file of the system (DOF, link mass and inertia properties, joints and constraints) and produces a series of C or Fortran files providing many useful functions needed to deal with Eqn. 2.12. Specifically we can read mass matrix, centripetal, gravity and centrifugal

\[ \text{http://www.sdfast.com/} \]
forces, kinematic positions and velocities and also inverse dynamics. One can also give desired accelerations and calculate required torques to derive the system using SD/FAST. It has actuation functions where user defines input force profiles regarding the state of the system which could be used to simulate the system forward. This simulation property is useful in case of manipulator robotics as they have fixed base. However in case of mobile and specifically legged robotics, dealing with hybrid states is rather difficult in SD/FAST. In our project, we have a model of the robot created by SD/FAST, using the license of our partners in Carnegie Mellon University (Prof. Christopher G. Atkeson).

3.3.2 CVXGEN

As part of our control method described later, we need a fast library to solve a convex quadratic problem (QP) with several convex equality and inequality constraints. CVXGEN\(^8\) is a fast library to solve such problems and it could be easily integrated with our controller. Advantages of using CVXGEN are:

- Fast integration: One can actually formulate his problem in an interface available on its website and the library therefore generates a couple of C codes that could be included in our controller which is written in cpp.

- Easy problem formulation: Although quadratic problems are quite formal, but one can define many other forms of convex functions with CVXGEN. For example we can exclude some variables from the objective function by simply not mentioning it. This is equal to setting the corresponding element of a matrix to zero.

- Sparse computation: The big advantage of CVXGEN which makes it very fast is the fact that it avoids unnecessary zero multiplications. In the formal definition of a quadratic problem, usually most of the elements in the matrices are zero, including our problem as well. Thus we can easily avoid these computations by formulating our problem efficiently in CVXGEN and then, it will convert the problem to the formal form implicitly, be generating C codes which basically have necessary multiplications written by individual variables, not big matrices. So computations in CVXGEN are not matrix-wise, rather they are tailored to the specific problem defined.

\(^8\)http://cvxgen.com/
3.4 Perception

So far, we explained all the packages and libraries used in either forward simulator or our controller. There is another important concept that we would rather explain here which is perception algorithm. In mobile robotics always, the problem of locating a robot in the surrounding environment is challenging. There are various methods and technologies introduced for such problem indoor and outdoor. Note the fact that positioning is tightly coupled with perception. The only difference is that perception mostly refers to creating a model of the environment while positioning is mainly determining the robot’s position in that environment. For the purpose of this project, we do not reach the point where a complex perception of the environment is required. Rather we would like to determine the position of the robot and especially its contacts locally which are crucial variables required for low-level control of the robot.

Among various advanced sensors on the robot, we use contact forces, IMU on the pelvis and joint position and velocity sensors. Other sensors like cameras and laser scanners need advanced data processing used to perceive environment and they are beyond the scope of this project. Based on low level sensors mentioned, we propose a simple odometry method which determines the position of the robot mostly locally which is important for the low level controller proposed in next chapter. We also build a hybrid odometry algorithm which then integrates foot positions over time to determine the position of the robot in general coordinate. Certainly, it suffers from the drift problem, but it is enough for planning few future steps. However contact forces are used to determine hybrid states, i.e. single support leg, flight or double support.

3.4.1 Constraint consistent local position

Remember constraints of Eqn 2.18 where position $x_c$ and velocity $\dot{x}_c$ of foot should be consistent with the assumption of having the foot in complete contact with the ground:

$$
\begin{align*}
    x_c &= f(q) = \text{const} \\
    \dot{x}_c &= J_c \dot{q} = 0 \\
    \ddot{x}_c &= J_c \ddot{q} + \dot{J}_c \dot{q} = 0
\end{align*}
$$

The first line is in fact 6 equations imposed by ground when the foot is in full contact. Other lines are derivatives which should be zero. $J_c$ is the Jacobian of constraints explained before and Cartesian positions relate to the center of foot polygon. From IMU data we can determine the posture of pelvis. Note that in our SD/FAST model of the robot, the reference frame is attached to the pelvis.
it pelvis frame hereafter). General coordinates of the robot are then determined by the location and orientation of this frame with respect to the origin. To determine the local position and posture of the robot we take the following steps at each time-step:

1. We place pelvis frame at \([0, 0, 0]\) and orient it with respect to IMU data.

2. We set joint positions and velocities in the model to be the same as position and velocity sensor readings from the robot.

3. We read positions of feet from the model and vertical contact forces from the robot.

4. If some force is zero and the corresponding position of the foot is upper than the other one, the foot is detected to be in swing mode.

5. Based on previous comparison, we can determine if the robot is in left-support, right-support or double-support (assuming there is no flight mode, although we can detect it with contact forces).

6. Biased to the left foot in double-support phase and assuming full foot contact, we measure Cartesian position and velocity of contacting foot from the model again (\(\text{FK}(left)\) or \(\text{FK}(right)\) where \(\text{FK}\) denotes forward kinematic).

7. We shift pelvis frame such that the position of contacting foot becomes \([0, 0, 0]\). We also set velocity of pelvis frame to be negative of contacting foot velocity. Thus foot velocity becomes zero in general coordinate frame. Note that posture data purely comes from IMU.

With the above-mentioned procedure, a local positioning of the robot is achieved which always assumes that the world’s origin is placed at the center of contacting foot. With the above formulation, \(x_{c, \text{pos}} = [0, 0, 0]\) and velocity \(\dot{x}_{c, \text{pos}} = [0, 0, 0]\).

### 3.4.2 Incremental odometry

In this project we do not have running gaits which include flight phase. Based on this assumption, we can determine the position of the swing foot right at touch down moment and incrementally, determine the global position of the robot. More specifically we define the following variables at the beginning of the program:

\[
P_{l, \text{freeze}} = [0, 0.13, 0] \quad \text{(3.2)}
\]

\[
P_{r, \text{freeze}} = [0, -0.13, 0]
\]
which denote initial positions of centers for each foot where dimensions are in meter. These variables are then updated right after phase transition moments and kept constant when no transition occurs. We then define the variable $odo$ as:

$$odo = P_{freeze} - P_{Inst.}$$  \hspace{1cm} (3.3)$$

Where:

$$P_{freeze} = \begin{cases} 
P_{l,freeze} & \text{if left/double support} \\
P_{r,freeze} & \text{if right support} 
\end{cases}$$  \hspace{1cm} (3.4)$$

And $P_{Inst.}$ is determined from step 6 in previous section as:

$$P_{Inst.} = \begin{cases} 
FK(left foot) & \text{if left/double support} \\
FK(right foot) & \text{if right support} 
\end{cases}$$  \hspace{1cm} (3.5)$$

We then set Cartesian position $x_{pelvis}$ and velocity $\dot{x}_{pelvis}$ of the robot to :

$$x_{pelvis} = odo$$  \hspace{1cm} (3.6)$$

$$\dot{x}_{pelvis} = -\dot{P}_{Inst.}$$

Note that $FK$ is calculated given assumptions of step 1 in the procedure mentioned previously. Therefore Eqn.3.6 is in fact a shift to the model that brings constraint consistency.

Up to now, we have completely introduced the simulation platform and positioning method we use as the basis of controlling the robot. In this study, we keep the sensor requirements as minimal as possible. Adding other sensory information such as vision or laser data can highly facilitate positioning problem. However in this project, we aim at solving the problem in worst conditions when these data resources are not available, in other words the robot is blind. In our future works, we can use more data from other sensors and diffuse it in our odometry algorithm to compensate for the drift and obtain a more precise positioning algorithm. In next chapter we discuss the low level controller of the robot which is responsible for generating joint torques.
Chapter 4

Whole body optimization, generating joint torques

In this chapter, we formulate our own torque generating method. Having discussions of previous chapter in mind, we prefer to use a dynamic-model based method which includes all the information about the dynamical state of the robot and thus can predict system’s behavior in future more precisely. One can break the control problem down to different levels of abstraction. To design the walking controller, we will first design a block in this chapter that converts Cartesian accelerations to joint torques. Controlling a floating based robot with many degrees of freedom is difficult in joint space. Many researches find joint trajectories by offline optimizations which may be stable only locally. One is interested in giving task-space trajectories however to reduce problem size. We would also like to have pure torque control rather than position control of joints, as it requires again finding constraint consistent trajectories.

The low level controller we discuss in this chapter is responsible for converting Cartesian variables (accelerations) to joint variables (torques). On top of this low level controller, a planner determines Cartesian accelerations based on finite state machines, given desired foot locations. This planner is in fact a continuous time controller that reduces number of control variables. There is finally a navigator that determines desired foot locations based on locomotion direction and speed. This is the general structure, however we have some simple scenarios at the end of this chapter in which high level blocks are not existing.

As concluded in chapter 2, we would like to use the whole body optimization approach which generates proper joint torques without knowing joint trajectories and under various Cartesian constraints. To summarize, this approach optimizes over joint accelerations and contact forces given Cartesian motion of some points of the robot (CoM and feet). We can then find joint torques from equation of motion. Hereby, we formulate our problem for the specific type of robot under control.
Figure 4.1: Walking controller architecture. The low level controller is responsible for generating joint torques, given Cartesian accelerations. The planner generates these accelerations given desired foot locations. The navigator also determines these locations based on desired speed.

4.1 Designing low level controller

In this chapter, we will introduce the low level controller which is responsible for generating joint torques.

4.1.1 On-board joint controller in simulator

Remember the chapter 3 where we introduced the DRCSim package. This package has in fact emulated on-board controllers of the robot as well with some internal APIs. The robot publishes its state with the rate of 1KHz over ROS and a controller may subscribe to read these packets. Then the controller will generate proper joint commands to control the robot which have the form of Eqn. 1.2:

$$
\tau = \tau_{desired} + K_p(q_{desired} - q) + K_d(\dot{q}_{desired} - \dot{q}) + K_i \int (q_{desired} - q)
$$

4.1

So for each joint, the subscribed controller can give a feed-forward torque $\tau_{desired}$ and desired trajectories $q_{desired}$ and $\dot{q}_{desired}$ as well as feedback gains $K_p$, $K_d$ and $K_i$. 

41
4.1.2 Robot states, combined position/force control

Looking back to joint names depicted in Fig.3.1, within our SD/FAST model the robot’s state \( q \in \mathbb{R}^{35} \) has the following variables:

\[
q = \begin{bmatrix}
  x_p \in \mathbb{R}^3 \\
x_o \in \mathbb{R}^3 \\
q_{bn} \in \mathbb{R}^4 \\
q_{l.leg} \in \mathbb{R}^6 \\
q_{r.leg} \in \mathbb{R}^6 \\
q_{l.arm} \in \mathbb{R}^6 \\
q_{r.arm} \in \mathbb{R}^6 \\
w \in \mathbb{R}^1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x_p \\
x_o \\
q_{bn} \\
q_{l.leg} \\
q_{r.leg} \\
q_{l.arm} \\
q_{r.arm} \\
w
\end{bmatrix}
= \begin{bmatrix}
  x \\
y \\
z \\
q \cdot x \\
q \cdot y \\
q \cdot z \\
q_{bn} \\
q_{l.leg} \\
q_{r.leg} \\
q_{l.arm} \\
q_{r.arm} \\
w
\end{bmatrix}^T
\]

\[
\begin{bmatrix}
  \text{back}_l \\
\text{back}_r \\
\text{mby} \\
\text{back}_m \\
\text{neck}_a \\
\text{l.leg}_a \\
\text{l.leg}_l \\
\text{l.leg}_r \\
\text{l.arm}_a \\
\text{l.arm}_l \\
\text{l.arm}_r \\
w
\end{bmatrix}^T
\]

Where \( x_p \) and \( x_o \) are Cartesian position and posture (orientation) of the pelvis frame. Names of variables show to what joint they refer regarding Fig.3.1. The only variable to explain is \( q \) which denotes the posture of pelvis frame we discussed in perception part of chapter 3, described with quaternions. We can also introduce \( \dot{q} \in \mathbb{R}^{34}, \ddot{q} \in \mathbb{R}^{34} \) and \( \tau \in \mathbb{R}^{34} \) as:

\[
\dot{q} = \begin{bmatrix}
v \\
\omega \\
\dot{q}_{bn} \\
\dot{q}_{l.leg} \\
\dot{q}_{r.leg} \\
\dot{q}_{l.arm} \\
\dot{q}_{r.arm}
\end{bmatrix}, \quad \ddot{q} = \begin{bmatrix}
\alpha \\
\dot{\alpha} \\
\ddot{q}_{bn} \\
\ddot{q}_{l.leg} \\
\ddot{q}_{r.leg} \\
\ddot{q}_{l.arm} \\
\ddot{q}_{r.arm}
\end{bmatrix}, \quad \tau = \begin{bmatrix}
0 \\
0 \\
\tau_{bn} \\
\tau_{l.leg} \\
\tau_{r.leg} \\
\tau_{l.arm} \\
\tau_{r.arm}
\end{bmatrix}
\]

Here we do not use the selection matrix \( S \) in Eqn.2.12 for simplicity. Note that the dimension reduces from 35 to 34 as the last element of quaternion orientation is not needed. The way we control the robot and deal with on-board controller is as follows:

- Left and right leg are purely torque controlled, having \( K_p = 0, K_d = 0 \) and \( K_i = 0 \) for 12 joints of \( q_{l.leg} \) and \( q_{r.leg}. \)
- Use full on-board controller for back, neck, left and right arms. But setting their accelerations to 0. So \( \ddot{q}_{bn} = 0, \ddot{q}_{l.arm} = 0, \ddot{q}_{r.arm} = 0. \) We therefore give desired angles and velocities of zero to the on-board controller to track.

With the above-mentioned policy, the upper body of the robot remains fixed. However as described later, the low-level controller generates compensating torques.
for $h(q, \dot{q})$ in Eqn.2.12 regarding their zero accelerations. Using upper body joints like back joints and arms may help a real human to walk more naturally and energy-efficient. However in this work like many other researches on walking, we disable them intentionally to reduce the dimensions of our problem. In future works, we would like to include some of these joints to let a natural walking sequence evolve. With the above assumptions, the Eqn.2.12 for this robot becomes:

$$
\begin{bmatrix}
\alpha \\
\dot{q}_{l, \text{leg}} \\
\dot{q}_{r, \text{leg}} \\
\tau_{l, \text{arm}} \\
\tau_{r, \text{arm}}
\end{bmatrix}
\begin{bmatrix}
a \\
\alpha \\
0 \\
0 \\
0
\end{bmatrix}
+ h(q, \dot{q}) =
\begin{bmatrix}
0 \\
0 \\
\tau_{l, \text{leg}} \\
\tau_{r, \text{leg}} \\
\tau_{l, \text{arm}} \\
\tau_{r, \text{arm}}
\end{bmatrix}
+ J^T_C \lambda \tag{4.4}
$$

Where $\lambda \in \mathbb{R}^{12}$ denotes all constraint forces. This equation therefore becomes the main EoM of robot we use in our low level controller. For position controlled joints, we give constant positions, zero velocities and constant feedback gains all the time. We also apply gravity compensation feed-forward torques per joint which correspond to zero desired accelerations.

### 4.1.3 Abstracting the model to center of mass

As planning over future hybrid states (i.e. steps) is computationally expensive, we would like to have a more simplified model representing overall state of the robot where center of mass (CoM) is an ideal choice. So we want the low level controller to reduce the dimensions of the robot to CoM level so that we can plan walking in a more abstract level. At the same time, we want to control the posture of the robot and keep it straight. In [44], Stephens first optimizes $\lambda$ and then feeds it into the equation of motion to obtain $\tau$. Recalling Eqn.2.36 from previous chapter, he controls over $\dot{L}$ and uses a PD feedback on torso posture as:

$$
\dot{L} = M_{\text{torso}} = K_p(\theta_{\text{pos, des}} - \theta_{\text{pos}}) - K_d(\dot{\theta}_{\text{pos}}) \tag{4.5}
$$

Where the desired angular rate is $\dot{\theta}_{\text{pos, des}} = 0$. Mentioning Eqn.2.36 again:

$$
\begin{bmatrix}
D_1 \\
D_2
\end{bmatrix}
\lambda =
\begin{bmatrix}
m \ddot{C}_{\text{des}} + F_g \\
M_{\text{torso}}
\end{bmatrix} \tag{4.6}
$$

Where $D_1$ and $D_2$ are defined in Eqn.2.37, $\ddot{C}_{\text{des}}$ is desired center of mass translational acceleration, $m$ is total mass, $F_g$ is gravity force and $M_{\text{torso}}$ is posture control which has replaced $\dot{L}$ in Eqn.2.36. Stephens solves this equation for $\lambda$
by optimization which includes CoP and torque constraints and then using $\lambda$, he obtains $\tau$. The same approach of regulating momentum is taken by Herzog and Righetti in [46] while they solve a single optimization problem including all variables at the same time.

In our approach, since the back joint connects pelvis to torso (see Fig.3.1) and is fixed all the time, we assume that the postures of torso and pelvis are the same and since robot’s reference frame is connected to pelvis, we would like to regulate $x_o$ which appears in Eqn.4.2. What makes our approach different is that we have control over the robot’s posture by imposing control over pelvis/torso angular acceleration $\alpha$ instead, appearing in Eqn.4.3. This has the advantage that $\alpha$ is directly the angular acceleration of pelvis/torso. Restating $\lambda$ in the form of contact forces and torques:

$$\lambda = [F_L \ T_L \ F_R \ T_R]^T \tag{4.7}$$

Where $F$ is the vector of contact forces and $T$ is contact torques. Our control law becomes:

$$F_L + F_R = m(g + \ddot{C}_{des})$$

$$\alpha = K_p(\theta_{pos}^{des} - \theta_{pos}) - K_d(\dot{\theta}_{pos}) \tag{4.8}$$

Where $\alpha$ regulates posture. Remember that $\alpha$ is in fact the rotational acceleration of robot’s frame attached to pelvis, appearing in robots state vector of Eqn.4.3.

### 4.1.4 Swing foot accelerations

So far we talked about stance control. However in swing phase, one is interested to control positions of swing foot by giving desired accelerations. This is the same approach taken by Stephens in [44] as well. Note that we are in fact converting 6 joint variables per leg to 6 Cartesian variables which make the control problem easier. Swing trajectories are quite important to avoid obstacles for locomotion on rough terrains. Here we make a platform which enables us having exact position control over foot trajectories. The important point however is to be constraint consistent. Recall from Eqn.2.18 that position, velocity and accelerations of contacting foot are constrained by:

$$x_c = f(q) = \text{const} \tag{4.9}$$

$$\dot{x}_c = J_c \dot{q} = 0$$

$$\ddot{x}_c = J_c \ddot{q} + \dot{J_c} \dot{q} = 0$$

The odometry block described in chapter 3 is responsible for the first and second equations while the low level controller must generate accelerations satisfying the
third constraint. Here we concatenate Jacobians of left and right feet together by defining $J_e \in \mathbb{R}^{12 \times 34}$ as:

$$J_e = \begin{bmatrix} J_{e, \text{left}} \\ J_{e, \text{right}} \end{bmatrix}$$  \hspace{1cm} (4.10)

Where 12 correspond to two feet each having 6 constraints and 34 for total DoF seen in Eqn. 4.3. We also compose constraint accelerations by concatenating foot Cartesian accelerations as:

$$\ddot{x}_e = \begin{bmatrix} \ddot{x}_{e, \text{left}} \\ \ddot{x}_{e, \text{right}} \end{bmatrix}$$  \hspace{1cm} (4.11)

Therefore in the controller, we have to find joint accelerations being consistent with Cartesian foot accelerations as well. Note that if a foot is contacting with ground, its corresponding Cartesian acceleration becomes zero.

### 4.1.5 CoP, friction and joint torque constraints

As discussed before, we want to optimize over all variables at the same time similar to the approach taken by Herzog and Righetti in [46]. Note that in contrast, Stephens in [44] breaks the problem down into two parts. We then take advantage of state of art fast solvers like CVXGEN that can handle this optimization problem online.

#### CoP constraints

Remember the discussion we had in chapter 2 on ZMP control approach. One is interested to keep the CoP inside the polygon of support to prevent the foot from rolling. Note that if CoP falls out, the assumption of full foot contact becomes violated and we should recalculate orientations and CoP. We would like to remain always in full contact mode as the robot does not have toes and the simulation becomes unstable in an ODE based simulator with edge contacts. As denoted by Stephens in [44] as well, we can put constraints on contact torques for each foot by:

$$-w_f \leq \frac{T_x}{F_z} \leq w_f$$  \hspace{1cm} (4.12)

$$-l_f \leq \frac{T_y}{F_z} \leq l_f$$

Where foot polygon is assumed to be rectangle with width of $2w_f$ in $y$ direction and length of $2l_f$ in x direction. We can also mention them as linear constraint like:

$$-w_fF_z \leq T_x \leq w_fF_z$$  \hspace{1cm} (4.13)

$$-l_fF_z \leq T_y \leq l_fF_z$$
Note that in double support phase, although one can define a total CoP for both feet, the assumption of having two individual CoPs does not violate any physical rule. Because CoP (not ZMP) is merely depending on a single contact’s ankle joint and normal force.

**Friction constraints**

For each foot, we would like to not violate the friction cone as it is important for stability. One can relate friction forces with surface normal forces by:

$$|\sqrt{\frac{F_x^2 + F_y^2}{F_z}}| \leq \mu \quad (4.14)$$

Which is not convex. Rather we can decouple $F_x$ and $F_y$ by:

$$\frac{|F_x|}{F_z} \leq \frac{\mu}{\sqrt{2}}$$

$$\frac{|F_y|}{F_z} \leq \frac{\mu}{\sqrt{2}} \quad (4.15)$$

Which could be written in forms of linear inequalities recognizable by CVXGEN. We can add rotational constraints as well as contact constraints like [45] by:

$$-\frac{\mu}{\sqrt{2}} F_z \leq F_x \leq \frac{\mu}{\sqrt{2}} F_z$$

$$-\frac{\mu}{\sqrt{2}} F_z \leq F_y \leq \frac{\mu}{\sqrt{2}} F_z$$

$$-\mu_R F_z \leq T_z \leq \mu_R F_z$$

$$0 \leq F_z$$

Where $\mu_R$ is rotational friction. We can therefore add all CoP and friction constraint together to have:

$$-\begin{bmatrix} \frac{\mu}{\sqrt{2}} \\ \frac{\mu}{\sqrt{2}} \\ 1 \\ w_f \\ l_f \\ \mu_R \end{bmatrix} F_z \leq \begin{bmatrix} F_x \\ F_y \\ 0 \\ T_x \\ T_y \\ T_z \end{bmatrix} \leq \begin{bmatrix} \frac{\mu}{\sqrt{2}} \\ \frac{\mu}{\sqrt{2}} \\ 1 \\ w_f \\ l_f \\ \mu_R \end{bmatrix} F_z \quad or \quad -\lambda_{lim} F_z \leq \begin{bmatrix} F_x \\ F_y \\ 0 \\ T_x \\ T_y \\ T_z \end{bmatrix} \leq \lambda_{lim} F_z \quad (4.17)$$
Joint torque limits

For actuator torques, we have two different constraints, lower/upper limits as:

\[-\tau_{\text{min}} \leq \tau_{\text{des}} \leq \tau_{\text{max}}\]  \hspace{1cm} (4.18)

and rate limits:

\[-d\tau_{\text{min}} \leq \dot{\tau}_{\text{des}} \leq d\tau_{\text{max}}\]  \hspace{1cm} (4.19)

Where actuators are assumed to be symmetric, i.e. \(\tau_{\text{min}} = \tau_{\text{max}}\) and \(d\tau_{\text{min}} = d\tau_{\text{max}}\). Considering actual control time-step determined by controller command publishing rate, we make the rate constraints recognizable by CVXGEN as:

\[-\tau_{\text{max},i} \leq \tau_{\text{des},i} \leq \tau_{\text{max},i}\]  \hspace{1cm} (4.20)

For each joint. One can therefore combine these inequalities to make the optimization problem simpler by:

\[\tau_{\text{down}} \leq \tau_{\text{des},i} \leq \tau_{\text{up}}\]  \hspace{1cm} (4.21)

\[\tau_{\text{down}} := \max(-\tau_{\text{max},i}, -\Delta\tau_{\text{max},i} + \tau_{\text{prev},i})\]

\[\tau_{\text{up}} := \min(\tau_{\text{max},i}, \Delta\tau_{\text{max},i} + \tau_{\text{prev},i})\]

Although as will be described later, we do not explicitly optimize joint torques, we can write them in terms of accelerations and contact forces using EoM and we can readily add these joint torque limit constraints by linear inequalities to the system. Note that \(\tau_{\text{prev}}\) is the command generated in previous time-step.

Existence of constraint

All previous constraints are defined when a foot is in contact with the ground. If not, we should force all 6 variables in \(F\) and \(T\) appearing in Eqn.4.7 to zero. Therefore we define the two following boolean variables:

\[i_{\text{left}} = \begin{cases} 1 & \text{if left/double support} \\ 0 & \text{else} \end{cases}\]  \hspace{1cm} (4.22)

\[i_{\text{right}} = \begin{cases} 1 & \text{if right/double support} \\ 0 & \text{else} \end{cases}\]

We then add these constraints to the optimization:

\[(1 - i_{\text{left}}) \begin{bmatrix} F_L \\ T_L \end{bmatrix} = 0_{6 \times 1}\]  \hspace{1cm} (4.23)

\[(1 - i_{\text{right}}) \begin{bmatrix} F_R \\ T_R \end{bmatrix} = 0_{6 \times 1}\]
So if a foot becomes out of contact, it forces corresponding constraint forces and torques to zero. These zeros are then consistent with all other CoP and friction constraints.

4.1.6 Formulating final quadratic problem

So far we discussed all the equality and inequality constraints that should be considered while doing optimizations. Note that as it appears in Eqn.4.4, we do not need to include all the mass matrix and corresponding parts in the optimization. Rather we only choose blocks corresponding to nonzero elements of vectors. More specifically, defining \( \ddot{q}_{\text{leg}} \in \mathbb{R}^{12}, \tau_{\text{leg}} \in \mathbb{R}^{12} \) and \( \tau_{\text{arm}} \in \mathbb{R}^{12} \) we have:

\[
\begin{bmatrix}
A_{1e} & A_{2e} & X & B_e & X \\
X & X & X & X & X \\
A_{1i} & A_{2i} & X & B_i & X \\
X & X & X & X & X
\end{bmatrix}
\begin{bmatrix}
a \\
\alpha_{\text{des}} \\
0 \\
\ddot{\ddot{q}}_{\text{leg}} \\
0
\end{bmatrix}
+ \begin{bmatrix}
H_e \\
X \\
H_i \\
X
\end{bmatrix} + \begin{bmatrix}
0 \\
\tau_{\text{om}} \\
\tau_{\text{leg}} \\
\tau_{\text{arm}}
\end{bmatrix} \lambda = \begin{bmatrix}
\ddot{\alpha}_{\text{des}} \\
0 \\
\ddot{\ddot{q}}_{\text{leg}} \\
0
\end{bmatrix}
\tag{4.24}
\]

Where \( A_{1e}, A_{2e}, B_e, H_e \) and \( C_e \) matrices have 6 rows and \( A_{1i}, A_{2i}, B_i, H_i \) and \( C_i \) have 12 rows. \( X \) also denotes irrelevant elements. We also have the foot acceleration constraints as:

\[
\begin{bmatrix} A_{1c} & A_{2c} & X & B_c & X \end{bmatrix}
\begin{bmatrix} a \\
\alpha_{\text{des}} \\
0 \\
\ddot{\ddot{q}}_{\text{leg}} \\
0 \end{bmatrix} + \dot{J}_c \dot{q} = \begin{bmatrix} \dddot{x}_{\text{c.left}} \\
\dddot{x}_{\text{c.right}} \end{bmatrix}
\tag{4.25}
\]

To put everything together, the following diagram shows inputs, outputs, objective function and constraints if this optimization:
\[
\begin{bmatrix}
\dot{C}_{\text{des}} & \alpha_{\text{des}}
\end{bmatrix} \in \mathbb{R}^6, \quad \dot{x}_{\text{c.left}} \in \mathbb{R}^6, \quad \dot{x}_{\text{c.right}} \in \mathbb{R}^6, \quad \text{state} \in \mathbb{N}
\]

\[\begin{align*}
\Downarrow
\end{align*}\]

\[
\begin{align*}
\min_{a, \ddot{q}_{\text{des}}, \lambda} & \quad a^T Q_a a + \ddot{q}_{\text{des}}^T Q_{\dot{q}} \ddot{q}_{\text{des}} + \lambda^T Q_{\dot{\lambda}} \lambda + R_a a + R_{\dot{q}} \ddot{q}_{\text{des}} + R_{\dot{\lambda}} \lambda \\
\text{s.t.} & \quad (A_{1e}) a + (A_{2e}) \alpha_{\text{des}} + (B_e) \dddot{q}_{\text{des}} + H_e = (C_e) \lambda \\
& \quad (A_{1e}) a + (A_{2e}) \alpha_{\text{des}} + (B_c) \dddot{q}_{\text{des}} + \dot{J}_c q = \begin{bmatrix} x_{\text{c.left}}^T & \dot{x}_{\text{c.right}}^T \end{bmatrix}^T \\
& \quad [F_x \ F_y \ F_z]_L^T + [F_x \ F_y \ F_z]_R^T = m (g + \ddot{C}_{\text{des}}) \\
& \quad (1 - i_{\text{left}}) [F_x \ F_y \ F_z \ T_x \ T_y \ T_z]_L^T = 0_{6 \times 1} \\
& \quad (1 - i_{\text{right}}) [F_x \ F_y \ F_z \ T_x \ T_y \ T_z]_R^T = 0_{6 \times 1} \\
& \quad \tau_{\text{down}} \leq (A_{1i}) a + (A_{2i}) \alpha_{\text{des}} + (B_i) \dddot{q}_{\text{des}} + H_i - (C_i) \lambda \leq \tau_{\text{up}} \\
& \quad -\lambda_{\text{lim}} F_{z,L} \leq [F_x \ F_y \ 0 \ T_x \ T_y \ T_z]_L^T \leq \lambda_{\text{lim}} F_{z,L} \\
& \quad -\lambda_{\text{lim}} F_{z,R} \leq [F_x \ F_y \ 0 \ T_x \ T_y \ T_z]_R^T \leq \lambda_{\text{lim}} F_{z,R}
\end{align*}
\]

With variables defined as:

- \(Q_a \in \mathbb{R}^{3 \times 3}, Q_{\dot{q}} \in \mathbb{R}^{12 \times 12}, Q_{\dot{\lambda}} \in \mathbb{R}^{12 \times 12}\), all positive semi-definite (psd)
- \(R_a \in \mathbb{R}^{1 \times 3}, R_{\dot{q}} \in \mathbb{R}^{1 \times 12}, R_{\dot{\lambda}} \in \mathbb{R}^{1 \times 12}\), all having positive elements
- \(A_{1e}, A_{2e}, B_e, H_e, C_e\) : defined in Eqn 4.24
- \(A_{1i}, A_{2i}, B_i, H_i, C_i\) : defined in Eqn 4.24
- \(A_{1c}, A_{2c}, B_c\) : defined in Eqn 4.25
- \(\tau_{\text{down}}, \tau_{\text{up}}\) : defined in Eqn 4.21
- \(\lambda_{\text{lim}}\) : defined in Eqn 4.17
- \(\text{state}\) : is planner’s hybrid state in Fig 4.1. We describe this input variable later.
- \(i_{\text{left}}, i_{\text{right}}\) : defined in Eqn 4.23 based on \(\text{state}\) variable.
• $m$ : total mass
• $g$ : gravity vector

And note that $\lambda$ is:

$$\lambda = \begin{bmatrix} F_x & F_y & F_z & T_x & T_y & T_z \end{bmatrix}_L^T \begin{bmatrix} F_x & F_y & F_z & T_x & T_y & T_z \end{bmatrix}_R^T \quad (4.27)$$

Where $F$ denotes force and $T$ denotes torque. The choice of $Q_a$, $Q_q$ and $Q_\lambda$ positive semi-definite matrices is quite arbitrary. Note that for this robot, we have $n = 28$ as number of joints, 6 Cartesian degrees of freedom in 3D space and $p = n$ as the number of actuated joints. In single support phase either on left or right, we have $k = 6$ degrees of freedom whereas in double support we have $k = 12$. Recalling analysis of redundancy and over-actuation we had in projected inverse dynamics section, under dynamic-model based methods in chapter 2, the case for this robot is either fully actuated or over actuated. It means that in single support the system has single solution while in double support it can have infinite number of feasible solutions. With this quadratic objective function, we minimize norm of optimization variables. We will later comment on the distribution of contact forces which makes the solutions asymmetric. For the moment, $Q_a$, $Q_q$ and $Q_\lambda$ are all diagonal with equal elements and $R_a$, $R_q$ and $R_\lambda$ are again having equal elements.

Note that we can easily define the problem in CVXGEN and it automatically converts the problem to the standard formulation of a quadratic problem:

$$\min_x \quad x^T Q x + R x$$
$$\text{s.t.}$$
$$A x + B = 0$$
$$C x + D \geq 0$$

CVXGEN implicitly solves this problem and does not form matrices $Q$, $R$, $A$, $B$, $C$ and $D$, because most of elements in these matrices are zero and solving the problem formally becomes inefficient.

Having calculated $a$, $\ddot q_{\text{des}}$, $\lambda$, we can plug them in Eqn. (4.24) and obtain $\tau_{bn}$, $\tau_{\text{leg}}$, $\tau_{\text{arm}}$ by:

$$\begin{bmatrix} 0 \\ 0 \\ \tau_{bn} \\ \tau_{\text{leg}} \\ \tau_{\text{arm}} \end{bmatrix} = M(q) \begin{bmatrix} a \\ \alpha \\ 0 \\ \ddot q_{\text{leg}} \\ 0 \end{bmatrix} + h(q, \dot q) - J^T C \lambda \quad (4.29)$$

Which are feed-forward joint torques being sent to the on-board controller. Remember that for legs we have pure torque control while for back, neck and arms we have position control added to this feed-forward torque, realizing zero accelerations and compensating gravity.
4.1.7 Contact force distribution

Ideally, one would expect that in double support phase, contact forces would be fairly distributed between two contacts. It means that we expect a foot to tolerate more weight if CoM is closer to it. This does not happen however, because the optimizer is blind to these effects and it equally distributes forces between the two contacts. To solve this problem, we define another variable for the optimizer or low level controller block called $0 \leq \text{prop} = \frac{F_{z,L}}{F_{z,R}} < \infty$. This variable determines with what proportion forces should be distributed between contacts. $\text{prop} = 0$ corresponds to right leg support while $\text{prop} = \infty$ corresponds to left support. $\text{prop} = 1$ also means equal force distribution. The way we modify optimization parameters to include this effect is by modifying $Q_\lambda$ in Eqn. 4.26:

$$Q_\lambda = \text{diag}([Q_{\text{left}} \quad Q_{\text{right}}]) \quad (4.30)$$

$$Q_\lambda \in \mathbb{R}^{12 \times 12}, \quad Q_{\text{left}} \in \mathbb{R}^6, \quad Q_{\text{right}} \in \mathbb{R}^6$$

We then determine vectors $Q_{\text{left}}$ and $Q_{\text{right}}$ to be:

$$Q_{\text{left}} \sim \text{ones}(6) \times \frac{1}{1 + \text{prop}} \quad (4.31)$$

$$Q_{\text{right}} \sim \text{ones}(6) \times \frac{\text{prop}}{1 + \text{prop}}$$

This distribution ensures that at least, normal forces $F_z$ are distributed proportionally. Note that we apply proportion to the quadratic cost of all 6 constraint forces and torques and not just the normal force. This is because other 5 constraints are related to $F_z$ by Eqn. 4.17 and this may preserve symmetry. It means that if the normal force gets smaller for a leg, other forces become smaller too proportionally and they do not trigger inequality constraints of Eqn. 4.17.

As $\text{prop}$ is a variable given to the optimizer, one can determine this proportion arbitrarily while normally, $\text{prop}$ should show how CoM is close to one contact point and far from the other one. By default, similar to [45], we calculate $\text{prop}$ each time-step by:

$$\text{prop} = \frac{\hat{P}_{\text{CoM}} - P_{\text{right}}}{\hat{P}_{\text{CoM}} - P_{\text{left}}} \quad (4.32)$$

Where $\hat{P}_{\text{CoM}}$ indicates that the position is projected on the ground, i.e. $\hat{P}_{\text{CoM},z} = 0$.

4.1.8 Imposed state of the low level controller

The planner in Fig. 4.1 has to determine the state of the controller which is either left, right or double support. Note that from the viewpoint of planner and low
level controller, this state is the desired state, not the actual state of the robot. So in case of state mismatch, the robot produces zero contact force, while having its foot touching the ground and about to lift off. So given a single support phase for example, we force the variable prop to either 0 or $\infty$ and also determine booleans $if_{\text{left}}$ and $if_{\text{right}}$ variables in Eqn.4.23. With these variables, the optimizer will produce proper torques for different hybrid phases regarding expected existence of constraints.

4.1.9 Experiment setup

We will explain the value of parameters here in the Table 4.1 for our specific robot. Note that these values mostly correspond to physical properties of the robot and environment.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Eqn</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>90kg</td>
<td>4.8</td>
</tr>
<tr>
<td>$g$</td>
<td>$[0, 0, -9.8]$</td>
<td>4.8</td>
</tr>
<tr>
<td>$Q_{x}$, $Q_{q}$</td>
<td>diagonal elements = 10</td>
<td>4.26</td>
</tr>
<tr>
<td>$Q_{\text{left}}$</td>
<td>diagonal elements = $\frac{20}{1+\text{prop}}$</td>
<td>4.30</td>
</tr>
<tr>
<td>$Q_{\text{right}}$</td>
<td>diagonal elements = $\frac{20\times\text{prop}}{1+\text{prop}}$</td>
<td>4.30</td>
</tr>
<tr>
<td>$R_{x}$, $R_{q}$, $R_{\lambda}$</td>
<td>elements = 1</td>
<td>4.26</td>
</tr>
<tr>
<td>$\tau_{\text{max}}$</td>
<td>$[110, 180, 260, 220, 220, 90]$ N.m (per leg)</td>
<td>4.21</td>
</tr>
<tr>
<td>$\Delta\tau_{\text{max}}$</td>
<td>50 N.m (for all actuators)</td>
<td>4.21</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.5</td>
<td>4.16</td>
</tr>
<tr>
<td>$\mu_R$</td>
<td>0.5</td>
<td>4.16</td>
</tr>
<tr>
<td>$w_f$</td>
<td>6cm</td>
<td>4.12</td>
</tr>
<tr>
<td>$l_f$</td>
<td>12cm</td>
<td>4.12</td>
</tr>
</tbody>
</table>

prop is determined in Eqn.4.32. For the choice of $\Delta\tau_{\text{max}}$, we assume that $d\tau/dt = 5$ N.m/ms for each actuator and take the worst control rate of 10 ms which yields $\Delta\tau_{\text{max}} = 50$ N.m. Note that there were no data-sheet available for Atlas robot to make these values realistic. For the moment we chose some reasonable values and optimization platform is flexible if changes are required later.

4.1.10 Posture control

Since in all simulation scenarios in this project, we have a similar control policy over postures, we would rather mention it separately here. Looking back to Eqn.4.26 the inputs to the algorithm are in fact $3 \times 6$ acceleration variable, 12 of which...
correspond to feet, 3 corresponding to CoM translation and 3 corresponding to pelvis posture. During all simulations, we have a similar policy to control postures of pelvis, right and left foot, total of 9 variables. The desired posture of course for all of them on level ground is \([0, 0, 0]\). Although one might consider changing postures due to local slope of the surface or to assimilate human motion more naturally, but for the scope of this work, we assume that:

- We have level-ground (zero slope)
- The robot is always in full foot contact
- It does not slip

Thus, it is better to keep posture of swinging foot \([0, 0, 0]\) so that to have an ideal touch down and we can read contact forces with minimal error. Decomposing inputs of Eqn. 4.26 into translational and rotational components we have:

\[
\ddot{x}_{c, \text{left}} = \begin{bmatrix} \ddot{P}_{\text{left}} \\ \dot{O}_{\text{left}} \end{bmatrix} \quad (4.33)
\]

\[
\ddot{x}_{c, \text{right}} = \begin{bmatrix} \ddot{P}_{\text{right}} \\ \dot{O}_{\text{right}} \end{bmatrix}
\]

Where \(P\) corresponds to position and \(O\) corresponds to postures. Then we have a similar control policy over all postures:

\[
\alpha_{\text{des}} = K_{\alpha}^{p}(0 - O_{\text{pelvis}}) + K_{\alpha}^{d}(0 - \dot{O}_{\text{pelvis}}) \quad (4.34)
\]

\[
\ddot{O}_{\text{left}} = K_{\text{acc}}^{p}(0 - O_{\text{lfoot}}) + K_{\text{acc}}^{d}(0 - \dot{O}_{\text{lfoot}})
\]

\[
\ddot{O}_{\text{right}} = K_{\text{acc}}^{p}(0 - O_{\text{rfoot}}) + K_{\text{acc}}^{d}(0 - \dot{O}_{\text{rfoot}})
\]

Values for different parameters in posture control are listed in Table 4.2.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(K_{\alpha}^{p})</td>
<td>10</td>
</tr>
<tr>
<td>(K_{\alpha}^{d})</td>
<td>3.3</td>
</tr>
<tr>
<td>(K_{p}^{\text{acc}})</td>
<td>50</td>
</tr>
<tr>
<td>(K_{d}^{\text{acc}})</td>
<td>6.6</td>
</tr>
</tbody>
</table>

Table 4.2: Values of feedback gains used in posture controllers. The feedback laws are tuned such that we achieve fast and stable posture control.
4.1.11 CoM acceleration regularization

So far, we have explained all parts of the low level controller. However we would like to regularize incoming accelerations so that they do not cause optimization procedure to violate constraints. This may happen for example when a large acceleration is expected for CoM which causes rolling of feet and leading to edge contact instead of full feet contact. Here, we truncate the incoming acceleration command based on Inverted Pendulum model with foot which will be described thoroughly in next chapter. One can truncate vertical acceleration ($\ddot{C}_{des,z} < 9.8 m/s^2$) regarding jumping effects. $x-y$ accelerations for CoM are also restricted by robot’s weight and friction coefficients. Thus we truncate $\ddot{C}_{des,x}$, $\ddot{C}_{des,y}$ and $\ddot{C}_{des,z}$ by:

$$\ddot{C}_{des,x,min} \leq \ddot{C}_{des,x} \leq \ddot{C}_{des,x,max}$$ (4.35)
$$\ddot{C}_{des,y,min} \leq \ddot{C}_{des,y} \leq \ddot{C}_{des,y,max}$$
$$-\mu g \leq \ddot{C}_{des,x} \leq \mu g$$
$$-\mu g \leq \ddot{C}_{des,y} \leq \mu g$$
$$\ddot{C}_{des,z} < g$$

Where:

$$\ddot{C}_{des,x,min} = \frac{g}{(1 + \text{prop})C_z}(\text{prop} \times (C_x - L_x - l_f) + (C_x - R_x - l_f))$$ (4.36)
$$\ddot{C}_{des,x,max} = \frac{g}{(1 + \text{prop})C_z}(\text{prop} \times (C_x - L_x + l_f) + (C_x - R_x + l_f))$$
$$\ddot{C}_{des,y,min} = \frac{g}{(1 + \text{prop})C_z}(\text{prop} \times (C_y - L_y - w_f) + (C_y - R_y - w_f))$$
$$\ddot{C}_{des,y,max} = \frac{g}{(1 + \text{prop})C_z}(\text{prop} \times (C_y - L_y + w_f) + (C_y - R_y + w_f))$$

$C$ is position of CoM, $L$ is position of left contact and $R$ is position of right contact. To have a briefly prove, we know that the acceleration of a linear inverted pendulum in 2D will be:

$$\ddot{x} = \frac{g}{z_0} (x - x_{\text{base}})$$ (4.37)

Where simplified symbol $x$ is position, $z_0$ is height and $x_{\text{base}}$ is the position of contact. The underlying assumption is that an actuator in the pendulum always keeps height of CoM constant at $z_0$, making equations linear. Another version of above formulation is derived (in [47] as well) when there could be ankle torque and support region of $\pm dx$, shown in Fig.4.2:

$$\ddot{x} = \frac{g}{z_0} (x - x_{\text{base,cop}})$$ (4.38)

$$x_{\text{base}} - dx \leq x_{\text{base,cop}} \leq x_{\text{base}} + dx$$
Assuming that region of support is not welded to the ground. For a 2D linear inverted pendulum we calculate the force $F$ in a leg as:

$$F = F \times \cos(\theta) = F \times \frac{z_0}{L} = mg \tag{4.39}$$

Where $L$ denotes length of pendulum and $\theta$ is its angle with $g$ vector, i.e. vertical direction. We can conclude for a mass with double bars that:

$$F_L + F_R = mg \tag{4.40}$$

Thus we have:

$$F_L = \frac{prop \times mg}{1 + prop} \tag{4.41}$$

$$F_R = \frac{mg}{1 + prop}$$

Accelerations are:

$$m\ddot{x} = F_L \left(\frac{x - x_{L,cop}}{L_L}\right) + F_R \left(\frac{x - x_{R,cop}}{L_R}\right) \tag{4.42}$$

We also know from Eqn.4.39 that:

$$F_L = \frac{F_L z_0}{L_L} \tag{4.43}$$

$$F_R = \frac{F_R z_0}{L_R}$$

Replacing $F_L$ and $F_R$ in the Eqn.4.42 with above expressions we have:

$$m\ddot{x} = F_L \frac{L_L (x - x_{L,cop})}{z_0} + F_R \frac{L_R (x - x_{R,cop})}{z_0} \tag{4.44}$$

55
Now replacing $F_{L,z}$ and $F_{R,z}$ in above equation with quantities of Eqn 4.41 we obtain:

$$m\ddot{x} = \frac{\text{prop} \times mg}{1 + \text{prop}} \frac{(x - x_{L,cop})}{z_0} + \frac{mg}{1 + \text{prop}} \frac{(x - x_{R,cop})}{z_0}$$ \hspace{1cm} (4.45)

And finally:

$$\dot{x} = \frac{g}{(1 + \text{prop})z_0} \left[ \text{prop}(x - x_{L,cop}) + (x - x_{R,cop}) \right]$$ \hspace{1cm} (4.46)

Considering:

$$x_{\text{base}} - dx \leq x_{\text{base,cop}} \leq x_{\text{base}} + dx$$ \hspace{1cm} (4.47)

We eventually have:

$$\ddot{x}_{\text{max,min}} = \frac{g}{(1 + \text{prop})z_0} [\text{prop}(x - x_{\text{base},L} \pm dx) + (x - x_{\text{base},R} \pm dx)]$$ \hspace{1cm} (4.48)

Which could be re-written with 3D variables to form Eqn 4.36. In this proof, we assume:

- Point mass for the body without inertia.
- No mass and inertia for leg.
- Two controllers together keep height of CoM constant.
- Leg contributions in tolerating the weight is represented by prop.

These simplifying assumptions although restrict accelerations, but help avoiding violation of constraints by input accelerations, coming to low level controller. We can make sure that rolling of feet do not happen after this regulation. We will describe LIPM with more details in next chapter.

### 4.1.12 Overall structure of low level controller

So far we were discussing various blocks in the low level controller. We had blocks determining force distribution proportion, acceleration regulation and posture control which were reducing problem dimension size and leaving only $3 \times 3$ parameters for the Planner block in Fig 4.1 to control which are $\dot{C}_{\text{des}}, \dot{P}_{\text{left}}$ and $\dot{P}_{\text{right}}$. It also has to impose the state of the controller. A brief schematic of low level controller is shown in Fig 4.3.

With this controller, we have actually abstracted a high dimensional problem in joint space to a low dimensional one in Cartesian space. The Planner block now only generates translational accelerations of 3 points in the robot and also imposes the state.
Figure 4.3: The low level controller appearing in Fig. 4.1. Inputs are 3 translational acceleration vectors and one state variable, coming from planner. Outputs are joint torques going to the simulator.

4.2 Testing low level controller

For testing functionality of low level controller, we have designed some specific planners in this section which perform simple stance and walking motions in order to show properties of the robot and low level controller’s performance.

4.2.1 Scenario 1: Squatting

Squatting refers to a movement in which the robot periodically sits down and stands up. A limiting factor though is the joint limit of \(uay\) which is an ankle joint with a pivot parallel to \(y\) axis. Refer to Atlas figure in Fig. 3.1 for a better view. To do squatting task, the following planner will be enough:

\[
\ddot{C}_{\text{des}} = K_p(P_{\text{des}} - P_{\text{CoM}}) + K_d(\dot{P}_{\text{des}} - \dot{P}_{\text{CoM}}) + \ddot{P}_{\text{des}} \quad (4.49)
\]
\[
\dot{P}_{\text{left}} = 0
\]
\[
\dot{P}_{\text{right}} = 0
\]

\text{state} = \text{Double Support}

1Watch movies at [http://biorob.epfl.ch/page-96274.html](http://biorob.epfl.ch/page-96274.html)
Figure 4.4: Squatting scenario. It includes vertical sinusoidal motion.

Where $K_p = 10$ and $K_d = 3.3$. We also define $P_{des}$ as:

$$P_{des} = \begin{bmatrix} 0 \\ 0 \\ 0.9 + A(-1 + \sin(\omega t + \phi)) \end{bmatrix}$$

(4.50)

$A = 0.05$, $\omega = \pi$, $\phi = \pi/2$ and 0.9 is maximum height. All dimensions are SI. $\dot{P}_{des}$ and $\ddot{P}_{des}$ are also obtained using analytical derivation of the above equation. With this simple planner, we expect the robot to sit down and stand up once every two seconds. The performance is shown in Fig.4.5.

In squatting test, we give a sinusoidal Cartesian position reference for height of CoM and the low level controller converts it to joint torques. Squatting is in fact a symmetric motion where legs have similar torques generated in joints. To quantify the limitations of this scenario, $A_{max} = 0.06$ and $\omega_{max} = 3\pi$ where we have the same tracking performance. Beyond that, the robot can not generate proper torques or it jumps off. In case of $\omega = 3\pi$, the contact force of a foot becomes 260N at minimum whereas it is 450N nominally regarding the weight of robot. Note that sinusoidal acceleration is proportional to square of $\omega$.

### 4.2.2 Scenario 2: Sagittal motion

In this section, we would like to investigate the effect of having a left/right movement to see how the proportion variable distributes forces between the two contacts.
Figure 4.5: Squatting simulation for testing the low level controller. The upper plot shows the desired and actual height of CoM where the controller is tracking the desired trajectory with good performance. The lower plot also shows torques generated in different joints of left leg for this motion.

Figure 4.6: Sagittal scenario. It includes left/right sinusoidal motion.
Here we use the same controller as Eqn.4.49, but with the following trajectories:

\[
P_{\text{des}} = \begin{bmatrix} 0 \\ \text{Asin}(\omega t + \phi) \\ 0.91 + Be^{-\frac{t^2}{\tau^2}} \end{bmatrix}
\]

\(A = 0.06, \omega = \pi, \phi = 0, B = -0.03m, \tau = 1s\) and 0.91 is in fact the initial height of CoM. All dimensions are SI. \(\dot{P}_{\text{des}}\) and \(\ddot{P}_{\text{des}}\) are also obtained using analytical derivation of the above equation. Forcing CoM to come down gradually for \(\dot{B}\) meters helps preventing legs to be over-extended, thus avoiding singularities. There is no need to decrease CoM’s height as much as squatting scenario since we do not have vertical motion and the risk of going into singularities. Note that squared exponentials have zero velocities in the beginning, making transitions smoother:

\[
\frac{d}{dt}e^{-\frac{t^2}{\tau^2}}|_{t=0} = -\frac{2t}{\tau^2}e^{-\frac{t^2}{\tau^2}}|_{t=0} = 0
\]

We would like to see how \(prop\) variable distributes contact forces between two contacts. Fig.4.7 depicts the output of this scenario.

The CoM controller has good tracking performance, even with low gains of \(K_p = 10\) and \(K_d = 3.3\) as in previous squatting test. Note the third plot in Fig.4.7, where quadratic costs are calculated based on Eqn.4.31 and Eqn.4.32. While CoM is bouncing between the two contacts, the low level controller gives more weight to the nearest contact foot. This proportion is not dependent on the frequency of bouncing and accelerations. It merely depends on relative positions. Since the height of CoM is almost kept constant in steady state, the summation of the two profiles in the forth plot of Fig.4.7 is constant while their values are proportional with:

\[
\frac{F_{\text{left},z}}{F_{\text{right},z}} \approx \frac{Q_{\text{right}}}{Q_{\text{left}}}
\]

Where \(Q_{\text{left}}\) and \(Q_{\text{right}}\) are quadratic costs in Eqn.4.31. Thus if for example \(Q_{\text{left}}\) is larger than \(Q_{\text{right}}\), it penalizes \(F_{\text{left},z}\) more and makes it smaller than \(F_{\text{right},z}\).

One might be interested to quantify maximum performance achievable in this scenario. With the same frequency, \(A_{max} = 0.09m\) beyond which, tracking performance decreases, i.e. the robot follows the sinusoid with delay and smaller variation, less then 0.09m. With the same \(A\) however, \(\omega_{max} = 1.2\pi\) because again, the acceleration is proportional to the square of \(\omega\) and the robot with its limit-sized feet can not provide such acceleration. This effect is shown in Fig.4.8 where the same
Figure 4.7: Sagittal motion simulation for testing the low level controller. The first plot shows the desired and actual height of CoM. The second plot shows position of CoM along y axis where the controller is tracking desired position. The third plot shows quadratic costs in Eqn. 4.31 for both legs. Since CoM goes left and right, prop variable changes and causes these costs to oscillate. In the last plot, contact forces are shown which depend on their quadratic cost in third plot.
Figure 4.8: The same sagittal motion, but with $\omega = 1.5\pi$. One can observe that due to saturation caused by acceleration regulator, the acceleration can not be arbitrarily large and thus, tracking performance is affected. The upper plot shows truncating effect and the lower one shows tracking performance.

scenario is run with higher frequency of $\omega = 1.5\pi$. Here the acceleration regulator shown in Fig.4.3 truncates acceleration which then affects tracking performance.

So far we discussed two scenarios where different aspects of low level controller were investigated. One is interested to know how the robot behaves in the existence of pushes while it is in stance mode.

4.2.3 Scenario 3: Push recovery and compliance test

In this scenario, we would like to investigate the effect of pushing the robot on the tracking performance of the low level controller. A compliant robot should adapt itself to the push and does not fail. While the robot is standing, we apply forces from different directions to the pelvis. The controller is the same as squatting controller in Eqn.4.49 with $K_p = 30$ and $K_d = 10$ and desired positions are:

\[
\mathbf{P}_{\text{des}} = \begin{bmatrix}
0 \\
0 \\
0.94 + Be^{-\frac{2}{\tau}}
\end{bmatrix}
\]  

(4.54)

Where $B = 0.07m$ and $\tau = 1s$. The strategy we use to maintain balance is to choose high gains for CoM position compared to low gains used for posture
controller inside low level controller. This causes the posture to have larger errors compared to CoM position. Note that we keep the low level controller intact. Maximum possible pushes are applied in Left, Front, Right and Back directions subsequently. Fig.4.10 shows details of this test.

There are a couple of points to be explained in this figure:

- Since posture controller is quite weak compared to CoM position controller, we can observe from the first plot in Fig.4.10 that it does not converge rapidly. The case is worse for posture around $y$ axis, since coronal (front, back) pushes are stronger.

- The summation of contact forces in the third and forth plots are nearly equal to the external force applied, verifying Newton’s law.

- We do not have force sensors to measure external forces like [44]. The emergence of non-zero planar forces at contact points is something automatic, i.e. compliance.

- Making the torso able to rotate helps keeping the CoM near origin. One can think of adapting arms or other joints for the same purpose.

- Larger gains do not necessarily make the robot withstanding larger pushes. Acceleration generated for CoM is limited by acceleration regulator block in Fig.4.3 and it depends physically on the polygon of support.

One may think that it should be possible to apply larger forces in $y$ direction (left, right) because of larger support polygon. In fact the choice of $prop$ restricts maximum sagittal pushes tolerable. If the robot was able to give more weight to
Figure 4.10: 4 maximal pushes applied in 5 seconds intervals and for duration of 3 second each: 50\(N\) to the left, 75\(N\) to the front, 50\(N\) to the right and 75\(N\) to the back. We can observe posture errors in first plot, CoM position errors in the second, contact forces in \(x\) direction in third and those in \(y\) direction in the last plot.
left leg for example, it could tolerate stronger pushes to the left. Note the force
distribution in Eqn.4.36 which could be re-written as:
\[
\ddot{C}_{\text{des},y}\min = \frac{g}{P_{\text{CoM},z}} \left[ (P_{\text{CoM},y} - w_f) - \frac{(\text{prop} \times P_{L,y} + P_{R,y})}{1 + \text{prop}} \right]
\]
\[
\ddot{C}_{\text{des},y}\max = \frac{g}{P_{\text{CoM},z}} \left[ (P_{\text{CoM},y} + w_f) - \frac{(\text{prop} \times P_{L,y} + P_{R,y})}{1 + \text{prop}} \right]
\]
Assuming \(\text{prop} \approx 1\) determined by CoM desired location, \(P_{\text{CoM},x} \approx 0, P_{\text{CoM},y} \approx 0\) and knowing the fact that \(P_{L,x} \approx 0, P_{L,y} \approx 0, P_{R,y} \approx -P_{R,y}\) we have:
\[
\dot{C}_{\text{des},y}\min = -\frac{g w_f}{P_{\text{CoM},z}}, \quad \dot{C}_{\text{des},y}\max = \frac{g w_f}{P_{\text{CoM},z}}
\]
\[
\dot{C}_{\text{des},x}\min = -\frac{g l_f}{P_{\text{CoM},z}}, \quad \dot{C}_{\text{des},x}\max = \frac{g l_f}{P_{\text{CoM},z}}
\]
Which verifies that we can apply larger force in front/back or x direction, since
\(l_f > w_f\) (foot has rectangular shape with \(\text{length} > \text{width}\)). Determining \(\text{prop}\) to
recover from strong pushes however needs sensing of external forces which is out of
scope of this research. Although in this chapter we recover from pushes statically
by posture change, we will use dynamic strategies like stepping in next chapters.

4.2.4 Scenario 4: Statically-Stable walking

Having discussed all aspects of the low level controller, how fast is the robot able
to do statically-stable walking? By this term, we refer to the policy of keeping
CoM inside support polygons all the time and stepping forward. To this end, we
propose the state machine shown in Fig.4.11.

Figure 4.11: Finite state machine used in static walk scenario. We have 3 states,
transition conditions and actions (performed right at transition).

Where variables and functions in Fig.4.11 are:

65
• $P_{\text{CoM}}, P_L$ and $P_R$: denote positions of CoM, left and right foot respectively.

• $P_{\text{CoM}, \text{des}}, P_{L, \text{des}}$ and $P_{R, \text{des}}$: denote desired positions of CoM, left and right foot respectively.

• $dir$: Variable indicating which foot will swing next.

• $P_{L, f}$ and $P_{R, f}$: are positions of feet saved right after touch-down. They help calculating next foot locations incrementally.

• $f(t, dir)$: is a shift function defined as:

$$
f(t, dir) = \begin{bmatrix}
A(1 - e^{-(t-t_0)^2/\tau^2}) \\
(0.13 - \alpha) \times \text{dir} \\
B \sin\left(\pi \frac{t-t_0}{T_{\text{timeout}}}\right)
\end{bmatrix}
$$

(4.57)

• $t_0$: is start time of a swing phase.

• $T_{\text{timeout}}$: is swing phase duration.

• $0.13$: is initial $y$ position of left foot, for right foot it is $-0.13 m$.

• $\alpha$: is a shift that makes foot closer.

• $A$: is the length of foot-step.

• $B$: is ground clearance.

For the center of mass also, we have:

$$
\begin{align*}
P_{\text{CoM, des}} &= P_{\text{base}} + (P_{\text{CoM}} - P_{\text{base}}) e^{-\frac{(t-t_0)^2}{\tau^2}} \\
P_{\text{base}} &= \begin{cases}
P_L & \text{dir} = 1 \\
P_R & \text{dir} = -1
\end{cases}
\end{align*}
$$

(4.58)

Which causes CoM bounce between $P_L$ and $P_R$ for subsequent steps. We produce control inputs for the low level controller by:

$$
\begin{align*}
\ddot{C}_{\text{des}} &= K_{p1}(P_{\text{CoM, des}} - P_{\text{CoM}}) + K_{d1}(\dot{P}_{\text{CoM, des}} - \dot{P}_{\text{CoM}}) + \ddot{P}_{\text{CoM, des}} \\
\ddot{P}_{\text{left}} &= K_{p2}(P_{L, \text{des}} - P_L) + K_{d2}(\dot{P}_{L, \text{des}} - \dot{P}_L) + \ddot{P}_{L, \text{des}} \\
\ddot{P}_{\text{right}} &= K_{p2}(P_{R, \text{des}} - P_R) + K_{d2}(\dot{P}_{R, \text{des}} - \dot{P}_R) + \ddot{P}_{R, \text{des}} \\
\text{state} &= \text{Left support, Double support or Right support}
\end{align*}
$$

(4.59)
Where first and second derivatives of desired trajectories are calculated analytically from Eqn.4.58 and Eqn.4.57. The proposed state machine in Fig.4.11 and its PD controllers give a sequence of motion to the robot. The strategy is that when the robot is in double support phase, the controller moves CoM to the polygon of next stance foot. When it reached the center of that polygon (within a small threshold $\delta R$), the opposite leg start swing phase and goes forward. During swing, CoM is still kept inside its polygon. After touch down, when the robots enters double support phase again, CoM moves to polygon of previously swinging leg. We use squared exponential transitions mentioned before to ensure smoothness of desired trajectories. To run the simulation, we choose parameters indicated in Table.4.3

Table 4.3: Values of parameters used in static walking planner.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0.4m</td>
</tr>
<tr>
<td>$T_{timeout}$</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0m</td>
</tr>
<tr>
<td>$B$</td>
<td>0.1m</td>
</tr>
<tr>
<td>$T_{timeout}$</td>
<td>1.0s</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2s</td>
</tr>
<tr>
<td>$C$</td>
<td>0.1m</td>
</tr>
<tr>
<td>$K_{p1}$</td>
<td>10</td>
</tr>
<tr>
<td>$K_{d1}$</td>
<td>3.3</td>
</tr>
<tr>
<td>$K_{p2}$</td>
<td>50</td>
</tr>
<tr>
<td>$K_{d2}$</td>
<td>16.6</td>
</tr>
<tr>
<td>$\delta R$</td>
<td>0.01m</td>
</tr>
</tbody>
</table>

The result of simulation using this parameter set is shown in Fig.4.12. The robot starts from stance mode and takes equal steps forward. Note that center of mass bounces between contacts quite largely to left and right. This is nature of static walking where one wants to make sure that CoM is always within some polygon and thus controllable. Considering Table.4.3, the timing parameters are quite slow and that is why the robot in Fig.4.12 goes about 2.5m in 50s (0.05m/s). This is again the nature of static walking and the fact that the support polygon is not large enough to perform fast motion.

Considering Table.4.3 and Eqn.4.57, the parameter to control this effect is $\alpha$. Choosing an $\alpha > 0$ causes feet to become closer, but one should also consider self collision which bounds $\alpha \leq 0.07m$. We simulated the robot with this bound, resulting in Fig.4.13. In this figure sagittal motions are less than Fig.4.12 and feet become closer.
Figure 4.12: The sequence of foot-steps generated by static walking controller. Note the large sagittal motion of CoM trajectory shown in red. It moves left and right to fall within the support polygon. Note that the support polygon in double support phase is the convex hull of both polygons.

Figure 4.13: The sequence of foot-steps generated by static walking controller with $\alpha = 0.07m$. Note foot-steps which become closer and thus, CoM sagittal motion reduces.
Note the PD controller performance in Fig. 4.13. Although it takes a small time to converge to center of polygon, but the controller is quite fast. It can also keep CoM fixed in its position during swing phase (within δR), even with relatively small gains of $K_{p1}$ and $K_{d1}$ in Table 4.3. This verifies that our low level controller has simplified the control problem precisely and provides acceptable tracking in task space to planner, i.e. upper control block.

There are various timing parameters in Table 4.3 which might make the walking faster. $T_{timeout}$ could be less, making swing phase shorter. $\tau_2$ depends on how fast the robot can perform sagittal motion, the analysis we had in previous tests. Maximum frequency $\omega_{max}$ was $1.2\pi$ which means a total time of about 1s from going left to right. Thus if we choose $\tau_2$ to be 0.5s for example, we will obtain Fig. 4.14.

![Figure 4.14](image)

Figure 4.14: The sequence of foot-steps generated by static walking controller with $\alpha = 0.07m$ and $\tau_2 = 0.5$. Compared to Fig. 4.13, PD controller’s performance is worse, but the robot can go further within the same timespan.

Note the difference between PD controller’s performance compared to Fig. 4.13. It is now worse because as soon as CoM falls inside $\delta R$ of center of polygon, swing phase starts but CoM has not yet settled. So PD controller makes the rest of its settling efforts during swing phase. However the robot is faster now, walking 3.5m in 50s (0.07m/s), but still the motion is very slow. Note that foot-step lengths are equal to the previous scenario.

One is interested to know if decreasing $T_{timeout}$ will make motion faster or not. In fact, the minimum of this variable could be 0.8s, with the same $\tau_2 = 0.5s$. This is mostly because the foot-step is large ($A = 0.4m$) and swing controller is not able to track it precisely in such short swing time. Note that $\delta R$ determines the threshold and could be increased, provided that CoM controller will settle during swing phase, though less stable.

The final important analysis is comparing accelerations between the last two scenarios where we change $\tau_2$. The comparison is shown in Fig. 4.15.

When we increase speed in sagittal plane, larger accelerations are produced by CoM controller in Fig. 4.15. Green vectors in this figure are scaled such that pink rectangles show acceleration limits imposed by acceleration regulator block in low
Figure 4.15: Comparison of accelerations between two scenarios with $\tau_2 = 2$ (Up) and $\tau_2 = 0.5$ (Down). From each CoM point, we draw a vector representing acceleration which is scaled by $P_{CoM,z}/g$. With this scale, the vector falls inside polygon, showing acceleration regulator’s limitations we had in the low level controller.

Note that at some moments in Fig.4.15, accelerations violate pink rectangles. These events are in fact happening in double support phase where the support polygon is the convex hull of both feet.

4.3 Conclusion

In this section, we formulated a low level controller and introduced different blocks used in this controller that help produce more stable results and simplify the problem. Our core optimizer consists of equality constraints regarding equation of motion and constraint consistent joint accelerations. It also has inequality constraints for friction, center of pressure and joint torque limits. The core optimizer therefore gets desired Cartesian acceleration of feet and CoM and generates feed-forward torques sent to the robot’s on-board controllers. Some of these torques generate motion while the rest are used for gravity compensation.

The low level controller has some additional blocks, making it more stable. It has a PD controller which maintains posture of the pelvis and feet upright. It limits incoming input accelerations to be CoP consistent and to avoid constraint violation in the optimizer. The force distribution is also determined based on the distance of CoM to foot locations. These blocks all together make a low level controller whose inputs are only translational accelerations of the three points (CoM and feet) and
desired hybrid state. It is similar to QP problems used in [44], [45] and [46]. The only difference is that they control torso posture by changing angular momentum, but this controller gets rotational acceleration of torso itself. Other advantages are:

- We only control leg joints in the robot which reduces dimensions of the problem.
- Our method is trajectory free, meaning that we do not need to know kinematic joint trajectories. So joints under control are purely torque controlled.
- The method is compliant, evident by its response to external pushes, even without sensing them.
- The low level controller reduces problem size considerably, making it easy to design planners for different tasks and motions, either stance or walking motions.
- Based on stance tests, tracking properties for CoM is acceptable in task space, even with relatively low gains of PD controllers in planner.
- Tracking is not as good for feet and one needs to choose higher gains in planner PD controllers. This is because we do not have access to closed form derivative of Jacobian matrix in our model library.
- Timings and details of delay analysis will be shown later, however low level controller block itself can solve the quadratic problem in 2ms on average in normal conditions. It also converges to a nearly optimal solution only in 6 iterations. However we let it continue to even 25 iterations with threshold of reaching optimality to be $1e^{-6}$.
- Although one can give bang bang accelerations to the optimizer and it generates continuous motion, but we have generated continuous Cartesian trajectories in our statically-stable walking test where low-gained PD controllers have to follow them.

The low level controller proposed in this section can make the robot walk with simple planners, but it is not enough for the requirements we had on fast walking and robustness. It is only responsible for compliance, robustness and joint torque limits. We would have a more comprehensive analysis on low level controller’s robustness to model errors. In next two chapters, we will introduce the concept of capture points for simple LIP models and propose planners which have various advantages over current statically-stable controller.
Chapter 5

Control Method 1: Captured walking

In previous chapter, we proposed a statically-stable walking method which was navigating center of mass to the center of contacting foot polygon and taking a conservative step. Then it was using both polygons in double support phase to move CoM to the opposite leg. This method was in fact fully based on having a support region within which we can control the robot. Although the robot was able to take large steps, but the CoM had a large sagittal motion to left and right which was resulting in unnatural walking. In this chapter, we want to improve our method by predicting the future motion of the robot. Specifically, we would like to predict what would happen if in single support or swing phase, the robot did not have any support region. With this prediction, it can start swing phase without having CoM close to center of polygon. Since a robot is like an inverted pendulum in swing phase, one is interested to know where CoM goes during this phase and where we could place the swing foot such that the accumulated energy in the system (increased linear momentum) can be captured.

To this end, we introduce the concept of linear pendulum mainly from [47] and [48]. Recall from previous chapter (4) that we introduced this model and formulated the problem in 2D. Here we formulate it in 3D and introduce the concept of capture point, although we will see later that it could be again, decoupled to 2D formulation along $x$ and $y$ axes.

5.1 3D Linear Inverted Pendulum Model (LIPM)

This model is in fact composed of a inertia-less mass and a mass-less leg. The difference between this model and ordinary inverted pendulum model is that we assume there is an actuator in the leg, keeping the height of mass constant. So
we do not linearize the model, rather we assume that we have enough control over CoM height in the low level controller, evident by tests in previous chapter. We mostly use this model during swing phase whereas in previous chapter, we used it in double support phase as well when regulating accelerations. Here in this chapter, we only use polygons during double support phase and we assume that the robot has point contact with ground in single support/swing phase. With this assumption we implicitly disable ankle joints. It means that giving LIPM accelerations to the low level controller during single support makes ankle torques nearly zero. However in reality, posture controller will need small torques to compensate for swing dynamics and posture error of pelvis.

5.1.1 LIPM with point ground contact

Fig 5.1 shows this model simply in 3D space. The EoM for this model is:

$$m \ddot{P} = F + mg$$  \hspace{1cm} (5.1)

Where \( P = [x, y, z] \), \( F = [f_x, f_y, f_z] \) and \( g = [0, 0, -9.8] \). Recall from Eqn 4.39 that the assumption of \( z_0 \) surface imposes \( F_z \) to be:

$$F_z = |F| \times \frac{z_0}{L} = mg$$  \hspace{1cm} (5.2)

Where \( L \) is current length of pendulum. Therefore one can calculate \( F \) by:

$$|F| = \frac{Lmg}{z_0}$$  \hspace{1cm} (5.3)
If we use the projection matrix $S$ defined as:

$$
S = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
$$

(5.4)

We can put $F$ in Eqn[5.1] and obtain:

$$
|S m \ddot{P}| = |S \frac{L mg}{z_0}|
$$

(5.5)

$$
\ddot{x} = \frac{L g}{z_0} \times \frac{x - x_{base}}{L}
$$

$$
\ddot{y} = \frac{L g}{z_0} \times \frac{y - y_{base}}{L}
$$

And finally:

$$
\ddot{x} = \frac{g}{z_0} (x - x_{base})
$$

(5.6)

$$
\ddot{y} = \frac{g}{z_0} (y - y_{base})
$$

So one can express accelerations linearly in terms of positions. Note that linear inverted pendulum controls CoM height and thus $z_0$ is assumed to be constant, enabling us to decouple EqM.

### 5.1.2 LIPM with support foot

As we saw in previous chapter, following the works [48] and [47], one can derive formulations for a more complete model of LIP. Here we assume that there is a finite sized support for the LIP, shown in Fig[5.2].

Similar equations could be written for this model as well, resulting in:

$$
m \ddot{x} = \frac{g}{z_0} (x - x_{base}) + \frac{\tau_{ankle,y}}{z_0} = \frac{g}{z_0} (x - x_{CoP})
$$

(5.7)

$$
m \ddot{y} = \frac{g}{z_0} (y - y_{base}) - \frac{\tau_{ankle,x}}{z_0} = \frac{g}{z_0} (y - y_{CoP})
$$

Here we can define $P_{CoP}$ as:

$$
P_{CoP} = P_{base} - \frac{1}{mg} \begin{bmatrix}
\tau_{ankle,y} \\
-\tau_{ankle,x} \\
0
\end{bmatrix}
$$

(5.8)

This $P_{CoP}$ is in fact the same as what we had before. Note that it is the responsibility of the low level controller to take care of these equations. Therefore
if we give accelerations obtained in Eqn. 5.6, inevitably, ankle torques become zero (Assuming inertia-less mass). Note also that $\tau_{\text{ankle}}$ has a component along $z$ axis which we are not concerned about, since this torque can not affect CoP and rolling effect:

$$
\tau_{\text{ankle},z} = -\frac{\tau_{\text{ankle},x}}{z_0}(x - x_{\text{base}}) - \frac{\tau_{\text{ankle},y}}{z_0}(y - y_{\text{base}})
$$  \hspace{1cm} (5.9)

Using accelerations obtained in Eqn. 5.6, this torque becomes zero as well.

### 5.1.3 LIPM with support foot and inertia mass

In this realistic model followed by [48] and [47], the mass has inertia similar to our robot, shown in Fig. 5.3.

The equation of motion for this system is:

$$
\begin{align*}
m\ddot{P} & = \mathbf{F} + mg \\
J\dot{\omega} & = \tau_{\text{hip}} - \omega \times (J\omega)
\end{align*}
$$  \hspace{1cm} (5.10)

Where $J$ is inertia matrix, $\omega = [\omega_x, \omega_y, \omega_z]$ is angular velocity and $\tau = [\tau_x, \tau_y, \tau_z]$ is hip torque.

In the same way, one can derive accelerations by:

$$
\begin{align*}
m\ddot{x} & = \frac{g}{z_0}(x - x_{\text{CoP}}) - \frac{\tau_{\text{hip},y}}{z_0} \\
m\ddot{y} & = \frac{g}{z_0}(y - y_{\text{CoP}}) + \frac{\tau_{\text{hip},x}}{z_0}
\end{align*}
$$  \hspace{1cm} (5.11)
In this equation, we have new terms corresponding to hip torques. These torques are actually hidden and generated by low level controller. In planner level when we want to produce translational accelerations, if we want $P_{CoP}$ to coincide with $P_{base}$, we should in fact assume hip torques to be zero. Remember that actual hip joint torques in the full robot with massed-links are not zero.  

Looking back to Eqn.5.10, we should consider $\omega$ and $\dot{\omega}$ to be zero. As we have formulated the low level controller in terms of pelvis posture and not angular momentum, we can not guarantee this especially in case of swing phase due to dynamics of swing leg. Therefore, we can not make $P_{CoP}$ and $P_{base}$ coincide ideally and thus, disable ankle torques and assume that we have point contact. However our assumptions are true for small ranges of motions unless the actual $P_{CoP}$ falls inside support polygon.

### 5.1.4 Instantaneous capture point

So far we have discussed various models of inverted pendulum and clarified with what assumptions in fact, we use the very simple LIPM without foot and inertia. In this section, we want to introduce the concept of capture point introduced by [47] and [17] which helps us absorbing the energy accumulated in the robot during swing phase. Later, we will motivate why we want to stick to the most simplified model of LIP.

The concept of instantaneous capture point could be described by making an important assumption: we assume that either the robot can instantaneously swing
the contacting foot and put it on another point on ground surface, or it has another massless/inertia-less leg that can be placed there whose swing dynamics does not affect equations of motion in Eqn. 5.6. Instantaneous capture point for LIPM is then defined as the point where if it becomes the new \( P_{\text{base}} \) it can capture all the energy so that the mass rests directly on top of this point with zero final velocity. The calculation of this point needs making equations dimensionless which we skip over and bring the result. Defining \( \omega_0 = \sqrt{\frac{2}{z_0}} \), we can write dimensionless orbital energies of the system (from [50]):

\[
\hat{E}_{\text{LIP},x} = \frac{1}{2} \left( \frac{x}{w_0 z_0} \right)^2 - \frac{1}{2} \left( \frac{x - x_{\text{base}}}{z_0} \right)
\]

\[
\hat{E}_{\text{LIP},y} = \frac{1}{2} \left( \frac{y}{w_0 z_0} \right)^2 - \frac{1}{2} \left( \frac{y - y_{\text{base}}}{z_0} \right)
\]

(5.12)

Where \( \hat{E}_{\text{LIP}} \) denotes dimensionless orbital energy, \( x \) and \( y \) denote mass position and \( x_{\text{base}} \) and \( y_{\text{base}} \) denote contact position. Based on these energies, one can distinguish three cases:

1. \( \hat{E}_{\text{LIP},x} > 0 \): \( x \) has enough energy to reach \( x_{\text{base}} \) and pass over.

2. \( \hat{E}_{\text{LIP},x} = 0 \): \( x \) reaches \( x_{\text{base}} \), but goes to rest, which is not stable though.

3. \( \hat{E}_{\text{LIP},x} < 0 \): \( x \) will not even reach \( x_{\text{base}} \) and changes its moving direction back before reaching \( x_{\text{base}} \).

We can conclude that the desired case is therefore \( \hat{E}_{\text{LIP}} = 0 \) where we could absorb all the energy. Solving these equations we obtain:

\[
P_{\text{ic}} = SP_{\text{CoM}} + \frac{SP_{\text{CoM}}}{\omega_0}
\]

(5.13)

Where \( S \) is projection matrix in Eqn. 5.4 and \( P_{\text{ic}} \) is the instantaneous capture point. \( P_{\text{CoM}} \) and \( P_{\text{CoM}} \) also denote CoM position and velocity. A more illustrative diagram is shown in Fig. 5.4.

One can write dynamics of \( P_{\text{ic}} \) as:

\[
\frac{P_{\text{ic}}}{\omega_0} = P_{\text{ic}} - P_{\text{base}}
\]

(5.14)

\[
P_{\text{ic}}(\Delta t) = [P_{\text{ic}}(0) - P_{\text{base}}]e^{\omega_0 \Delta t} + P_{\text{base}}
\]

So \( P_{\text{ic}} \) has a fast dynamics and rapidly goes away. In this chapter, we will propose a platform which controls CoM position and velocity so as to get the
expected $P_{ic}$ after swing phase close to next desired foot place. In this way we can assume that the energy is absorbed then.

In [47], authors have also defined a series of footsteps which cause the robot become capturable. Note that in reality, the swing distance is limited and the robot can not place its foot on arbitrary far locations. The strategy is to find necessary conditions to make a N-step capturable system, N-1 step capturable by taking a step. So assuming maximum distance of $l_{max}$ and minimum swing time of $T$, we should have:

\[ ||P_{ic}(T) - P_{base}|| \leq d_{N-1} + l_{max} \] (5.15)

Where $d_N$ denotes the maximum distance for which the robot is still N-step capturable. Combining this equation with Eqn.5.14, we have:

\[ ||P_{ic}(0) - P_{base}|| \leq (d_{N-1} + l_{max})e^{-\omega_0 T} = d_N \] (5.16)

Which leads to recursive expression of:

\[ d_N = (d_{N-1} + l_{max})e^{-\omega_0 T} \quad d_0 = 0 \] (5.17)

Therefore based on capture point position right before swing phase shown by $P_{ic}(0)$ in the equations, one can find out in how many steps, the system can rest. Followed by some calculations on this geometric series, one can obtain:

\[ d_\infty = l_{max} \frac{e^{-\omega_0 T}}{1 - e^{-\omega_0 T}} \] (5.18)

Using this variable, one can predict whether if the system is finally capturable or not. Regarding our feedback tunings and other parameters in the system, we

![Figure 5.4: Top view of inverted pendulum. Note the capture point which rapidly goes away along $P_{ic}$. (taken from [47])](image-url)
expect $l_{max} \leq 0.5m$. Choosing $T_{min} = 0.5s$, we can obtain $d_\infty = 0.14m$ which is enough for foot placement. This means that unless $\mathbf{P}_{ic}(0)$ does not go further than $0.14m$ of base foot ($||\mathbf{P}_{ic}(0) - \mathbf{P}_{base}|| \leq 0.14m$), the robot is capturable. Note that $\mathbf{P}_{ic}(0)$ is the capture point in double support phase, right before single support. In next section, we will design a walking control method based on capturability principle.

5.2 Captured walking using model predictive control

In this section, we will describe how we control the robot using the idea of capture points. Regarding the simplification we did for LIP model of the robot where we assumed no fly-wheel or inertia for the mass, we can not use lunging approaches taken by [49] for example for walking. In this approach the robot benefits from hip joint to lung and capture extra energy. Rather we propose multiple step planning to capture energy gradually.

This chapter consists of three parts. First, we introduce how we plan future steps using instantaneous capture points. Second, we will describe stance controller and finally we will propose our swing controller. Fig.5.5 shows the interconnection of different blocks. We determine next desired foot place $\mathbf{P}_{base,new}$ and desired capture point $\mathbf{P}_{ic}(0)$ in navigator and planner has to follow them.

![Figure 5.5: Combination of different blocks used to control the robot in this chapter.](image)

The total strategy of this controller is as follows:

- **Navigator:** In double support, we control $\mathbf{P}_{ic}(T)$ to fall close to the next desired foot place. $T$ is in fact the duration of single support phase. Note that we can calculate $\mathbf{P}_{ic}(0)$ in Eqn.5.14 because we know all the parameters and also $\mathbf{P}_{ic}(T)$ which is next desired foot place.

- **Stance planner:** In double support phase, we control CoM such that the system reaches desired $\mathbf{P}_{ic}(0)$. 79
• **Swing planner:** After reaching $P_{ic}(0)$, we start swing phase, moving the swing foot to the pre-determined desired foot place which coincides with $P_{ic}(T)$.

We will explain different parts respectively.

### 5.2.1 Navigator

First, we would like to describe a method which determines how hybrid states of the system evolve in future. This block is the navigator shown in Fig.4.1 which now determines a desired capture point for the robot to follow. Note that planner is in fact responsible for planning continuous states of the robot which usually has a finite state machine and specific laws in each state or phase. Planner can not plan over multiple hybrid states, the navigator is designed to do so instead.

In [51], authors use the concept of capture points in 3D on a 6 DoFs per leg robot using inverse kinematics. They produce patterns of future capture points which cause the robot stop after 5 steps. Based on these patterns, they obtain desired ZMP point for the robot which is then followed by a ZMP controller. To compare, we generate desired capture point similarly by solving its dynamics equation and assuming a fixed sequence of future steps. However our robot has a compliant low level controller compared to their inverse kinematics.

In our method, one should determine where the robot wants to put its foot first. Then the navigator finds a capture point for the robot adaptively by looking into future hybrid states, provided that conditions mentioned in previous chapter about capture point are not violated. The schematic of Fig.5.6 shows the 1-Step strategy where capture point coincides with next desired foot location and the robot rests in just one step.

![Figure 5.6: The schematic of 1-Step capturable controller that makes the robot resting at each foot-step. $P_{ic}$ is current capture point, $P_{ic}(0)$ is expected capture point right before swing, and $P_{ic}(T)$ is capture point right after swing which matches next foot-location. The stance planner has to move $P_{ic}$ towards $P_{ic}(0)$.](image)
In Fig. 5.6, the controller tries to match $P_{ic}$ to $P_{ic}(0)$ so that when swing starts, the system follows expected (falling) behavior. We measure matching again with a threshold $\delta R$, which is chosen to be $\delta R = 0.01m$. In this strategy, the robot has quite un-natural movement. The CoM goes left and right and has considerable sagittal motion. Remember the controller from previous section. The statically-stable walking controller was moving CoM even on top of center of support polygon which was very un-natural. The case is better here as we will show in experiments soon. But still this motion is not fast, specially in case that double support phase should actually have negligible time.

So far, what we explained 1-Step capturability. For point-wise contact LIP, 0-Step capturability region is a single point only. However for our robot with finite-sized foot, this region is larger. With this assumption, we can define a foot-step series where steps become smaller and the final $P_{ic}$ falls in the middle of the two feet. Thus the robot will go to the normal stance mode at the end. The idea can be visualized in Fig. 5.7.

Figure 5.7: N-Step capturable algorithm where N=5 here. We assume that after N steps, the system has a capture point in the middle of two feet and CoM rests there completely. Note that the system might rest at less than N steps too. But we assume more steps to have more natural movement and gradual deceleration.

Knowing foot locations, we can back-propagate final capture point and obtain desired $P_{ic}(0)$ as before. This procedure could be formulated by finding solutions of capture point dynamics in Eqn. 5.14 as follows:

$$\text{Forward: } H_f(P_{ic}(0), P_{base}) = P_{ic}(T) = (P_{ic}(0) - P_{base})e^{\omega_0 T} + P_{base} \quad (5.19)$$

$$\text{Backward: } H_b(P_{ic}(T), P_{base}) = P_{ic}(0) = (P_{ic}(T) - P_{base})e^{-\omega_0 T} + P_{base}$$

We can use these solutions obtained in [47] and plan for future by defining a series assuming fixed $T$:

$$P_{ic}[k+1] = H_f(P_{ic}[k], P_{base})$$
$$P_{ic}[k] = H_b(P_{ic}[k+1], P_{base}) \quad (5.20)$$
Now given $\mathbf{P}_{\text{base}}$ and $\mathbf{P}_{\text{swing}}$ in double support phase, where $\mathbf{P}_{\text{base}}$ will be the position of base when the robot goes to swing phase and $\mathbf{P}_{\text{swing}}$ is the position of swing foot, before swinging placed on the ground, we can define a series of $N$ foot-steps from $\mathbf{P}_{\text{base}}[0]$ to $\mathbf{P}_{\text{base}}[N-1]$ as:

$$
\begin{align*}
\mathbf{P}_{\text{base}}[0] &= \mathbf{P}_{\text{base}} \\
\mathbf{P}_{\text{base}}[k] &= \mathbf{P}_{\text{base}}[k-1] + \left[ \begin{array}{c} \text{step}_{\text{max}} \ast \frac{N-k}{N-1} \\
(-1)^k(\mathbf{P}_{\text{base,y}} - \hat{\mathbf{P}}_{\text{swing,y}}) \\
0 \end{array} \right]
\end{align*}
$$

Where $\text{step}_{\text{max}}$ is the length of first step and:

$$
\hat{\mathbf{P}}_{\text{swing,y}} = \text{saturate}(\mathbf{P}_{\text{swing,y}}, \alpha) \quad (5.22)
$$

Which truncates $y$ component of swing foot position to $\pm \alpha$ so that to make steps closer in $y$ direction and avoid large sagittal movements. Such sequence will have a form similar to Fig.5.7. The algorithm then gradually stops the robot by taking shorter steps. To find $\mathbf{P}_{\text{ic}}(0)$ over these $N$ steps, we should back-propagate the last capture point, falling in the middle of the two last foot locations. So we can write the total algorithm as:

$$
\begin{align*}
\mathbf{P}_{\text{ic}}[N] &= 0.5(\mathbf{P}_{\text{base}}[N-1] + \mathbf{P}_{\text{base}}[N-2]) \quad (5.23) \\
\text{for} & \quad k = N-1 : -1 : 0 \\
\mathbf{P}_{\text{ic}}[k] &= H_b(\mathbf{P}_{\text{ic}}[k+1], \mathbf{P}_{\text{base}}[k]) \\
\text{end}
\end{align*}
$$

$$
\begin{align*}
\mathbf{P}_{\text{ic}}(0) &= \mathbf{P}_{\text{ic}}[0] \\
\mathbf{P}_{\text{base,new}} &= \mathbf{P}_{\text{base}}[1]
\end{align*}
$$

Which determines the next foot step location and $\mathbf{P}_{\text{ic}}(0)$, the variables to be given to planner. So $N$ fixed steps in mind, if we assume that future double support phase take no time, the capture point will bounce between these steps and goes finally to $\mathbf{P}_{\text{ic}}[N]$. Note however that we can not expect zero-timed double supports in between for two reasons:

1. Disturbances may happen and stance planner has to compensate them.
2. We expect a desired average speed for the robot, not to rest finally.

So in practice, double support phases take time. From implementation point of view, the two navigation methods described before can be implemented easily using back-propagation for any $N \geq 1$. We can choose middle point as the last $\mathbf{P}_{\text{ic}}$ or we can choose $\mathbf{P}_{\text{base}}[N-1]$ instead to rest on foot and realize the first navigation with $N = 1$.  

82
5.2.2 Stance planner, model predictive control

\( \mathbf{P}_{ic}(0) \), the target capture point imposed by navigator which we want to follow, is in fact a combination of CoM position and its velocity. So we have infinite solutions with the same \( \mathbf{P}_{ic}(0) \). Thus if one uses a PD controller for example, because of intrinsic redundancy in capture points, he will not get smooth results. It takes considerable time for the system to match \( \mathbf{P}_{ic} \) with \( \mathbf{P}_{ic}(0) \), often with CoM moving backward which is not desired.

Authors in [51] use support polygon both in single support and double support phases to track desired capture points. However we only apply control in double support and let the robot fall in single support. The motivation is to avoid rolling or tilting of foot in single support when swing dynamics is also affecting CoP. Instead of ZMP control in [51], we directly control CoM during double support to retrieve the desired capture point. Rolling and tilting avoidance is done by our low level controller.

So we formulate a model predictive control (MPC) problem that uses discretization of the differential equation and predicts the CoM path for \( M \) future time-steps (i.e. over continuous-time states). The final point satisfies capture point constraint at the end and MPC in fact minimizes acceleration variations using null space of this constraint. We define:

\[
\begin{align*}
\mathbf{P} &= \begin{bmatrix} x \\ y \end{bmatrix} \\
\dot{\mathbf{P}} &= \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \\
u &= \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}
\end{align*}
\] (5.24)

for position, velocity and accelerations of CoM in \( z_0 \) surface (constant height) described before. \( \mathbf{u} \) is in fact the input of the system which we should calculate and give to the low level controller. Note that we can discretize this system by:

\[
\Delta t = \frac{T_h}{M} \]

(5.25)

\[
\begin{align*}
\mathbf{P}[k+1] &= \dot{\mathbf{P}}[k] \Delta t + \mathbf{P}[k] \\
\dot{\mathbf{P}}[k+1] &= \mathbf{u}[k] \Delta t + \ddot{\mathbf{P}}[k]
\end{align*}
\]

The MPC controller therefore starts working in double support phase and has to match current capture point of the system with a desired \( \mathbf{P}_{ic}(0) \). If we show start time of double support phase by \( t_0 \), nominal duration of this phase by \( T_{ds} \) and current simulation time by \( t \), we can calculate expected remaining time of the double support phase by:

\[
T_h = max(T_{ds} - (t - t_0), \delta t)
\] (5.26)

Where \( T_h \) is called horizon time and is at least \( \delta t \). The controller brings \( \mathbf{P}_{ic}(t) \) to \( \delta R \) neighborhood of \( \mathbf{P}_{ic}(0) \) in expected time \( T_{ds} \). If it could not do it to any
reason like large perturbation for example, we give a minimum horizon of $\delta t = 0.1s$ so that not to generate large accelerations. The choice of $\delta t$ does not affect stability. Because large accelerations are saturated by acceleration regulator block in low level controller. However choosing large $\delta t$ will slow down the settling time and not desired. This variable horizon strategy have better timing compared to fixed horizon. Since we want our final state $P$ and $\dot{P}$ to satisfy $P_{ic}(0)$, if we rather use a fixed horizon $T_h$:

- if $T_h < T_{ds}$, then MPC generates large accelerations specially if $T_h$ is very small.
- if $T_h > T_{ds}$, then MPC converges slowly and has the risk of overshoots and even backward speeds.

With our variable $T_h$, MPC will generate smooth accelerations and has better settling properties. Given initial states of the CoM ($P_0(t)$ and $\dot{P}_0(t)$) and final desired $P_{ic}(0)$, we formulate our MPC controller as follows:

$$\begin{align*}
P_0(t) \in \mathbb{R}^2, \quad \dot{P}_0(t) \in \mathbb{R}^2, \quad P_{ic}(0) \in \mathbb{R}^2, \quad \Delta t \in \mathbb{R}^1, \quad M \in \mathbb{N} \quad (5.27)
\end{align*}$$

The output of this optimization is $\mathbf{u}[1]$, the acceleration vector to be given to the low level controller for this time-step. So we run optimization again using CVXGEN per time-step in double support phase. The main constraint is to achieve $P_{ic}(0)$ at time $T_h$ while the quadratic problem minimizes variations in the acceleration profile.

Regarding Eqn. 4.36, although one can describe prop variable via CoM positions and support feet, it is not possible to obtain a linear inequality to limit accelerations.
within QP. We thus truncate accelerations with the regulator block described in previous chapter. The system may fail in high walking speeds, however it works for lower speeds without exceeding acceleration bounds, verified by simulations.

One can also express the convex hull of support polygons as a new polygon with linear inequality constraints for CoM position. This however does not make sense without previous constraint of acceleration limitation. If MPC plans a path near boarders of support polygon, this path is not valid. Since MPC does not know the fact that possible accelerations are low in these regions regarding LIP model with foot. We assume that start and end points of the path are within the polygon. Thus we expect MPC to plan a smooth paths inside, assuming uniform accelerations regarding the optimization objective function. The system will work in low speeds however and with small sagittal motions. If CoM goes to margins, it can not come back since there is no counter-acceleration.

Since the low level controller truncates accelerations which are not feasible and if for any reason CoM goes to the margins, there is no acceleration available to bring it back. Therefore we do not include this constraint in the optimization, again verifying by simulations that in lower speeds, CoM is controlled to desired state and remains inside support polygon.

The formulation of our MPC controller does not need any tuning except choice of $M$. This variable determines preciseness of trajectory prediction and we prefer to increase it as much as possible. Regarding CVXGEN limitations in code generation, $M = 50$ is chosen for all simulation scenarios in this chapter. So far we discussed planner policies in stance phase. Note that for feet, we give zero accelerations desired, since they are in contact with the ground.

### 5.2.3 Swing planner, LIPM accelerations

As discussed before, in swing phase we assume that there is no foot to provide ankle torques. However we saw that as our robot is described with an LIP having inertia, we might have $\mathbf{P}_{\text{CoP}}$ not coinciding with $\mathbf{P}_{\text{base}}$ due to hip torques. In swing phase, LIP accelerations are used for CoM as Eqn. 5.6. For stance foot in contact with ground, we give zero accelerations while for swing foot, we generate an arc trajectory towards the desired position.

We show current position of CoM by $\mathbf{P}_{\text{CoM}}$, current velocity by $\dot{\mathbf{P}}_{\text{CoM}}$ and contacting foot by $\mathbf{P}_{\text{base}}$. We also use the same PD controller like previous chapter.
for controlling CoM height. Planner accelerations are therefore:

\[
\ddot{C}_{\text{des}} = \begin{cases} 
\begin{bmatrix}
    u_x[1] \\
    u_y[1] \\
    K_p1(P_z - P_{\text{CoM},z}) + K_d1(\dot{P}_z - \dot{P}_{\text{CoM},z}) + \ddot{P}_z \\
    \omega_0^2(P_{\text{CoM},x} - P_{\text{base},x}) \\
    \omega_0^2(P_{\text{CoM},y} - P_{\text{base},y}) 
\end{bmatrix} & \text{if Double support} \\
0 & \text{else}
\end{cases}
\]

\(\ddot{P}_{\text{left}} = \begin{cases} 
K_p2(P_{L,\text{des}} - P_L) + K_d2(\dot{P}_{L,\text{des}} - \dot{P}_L) + \ddot{P}_{L,\text{des}} & \text{if Right support} \\
0 & \text{else}
\end{cases}\)

\(\ddot{P}_{\text{right}} = \begin{cases} 
K_p2(P_{R,\text{des}} - P_R) + K_d2(\dot{P}_{R,\text{des}} - \dot{P}_R) + \ddot{P}_{R,\text{des}} & \text{if Left support} \\
0 & \text{else}
\end{cases}\)

Where:

\[P_z = 0.94 + Ae^{-\frac{t^2}{\tau^2}}\]  

And \(z_0 = P_{\text{CoM},z}\). We use a similar finite state machine to Fig.4.11 for determining states of the system and controlling the robot shown in Fig.5.8. Most of variables and transition rules are like static walking, except that here we generate swing trajectories differently.

\[f(t, P_{\text{prev}}, P_{\text{new}}) = \begin{bmatrix}
P_{\text{prev},x}(e^{-\frac{(t-t_0)^2}{\tau^2}}) + P_{\text{new},x}(1 - e^{-\frac{(t-t_0)^2}{\tau^2}}) \\
P_{\text{prev},y}(e^{-\frac{(t-t_0)^2}{\tau^2}}) + P_{\text{new},y}(1 - e^{-\frac{(t-t_0)^2}{\tau^2}}) \\
B\sin(\pi\frac{t-t_0}{T_{ss}})
\end{bmatrix}\]  

Where \(T_{ss}\) is nominal single support duration time and \(t_0\) is start time of swing phase. This smooth transition function will generate an arc trajectory, bringing the

Figure 5.8: The finite state machine used in the planner.
foot from \( \mathbf{P}_{prev} \) to \( \mathbf{P}_{new} \). Note that \( \mathbf{P}_{new} \) is determined by navigator. Recall that the transition condition of matching capture points is in fact \( |\mathbf{P}_{ic} - \mathbf{P}_{ic}(0)| < \delta R \) where \( \delta R \). Throughout this chapter and simulations, we keep \( \delta R = 0.01m \) constant.

### 5.2.4 Parameter tuning

There are some parameters to be tuned in the continuous-time MPC controller we proposed in this chapter, planners and navigators. These variables are listed in Table 5.1.

Table 5.1: Values of the parameters used in captured walking controller of this chapter.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>X</td>
<td>Eqn 5.21</td>
</tr>
<tr>
<td>( \text{step}_{max} )</td>
<td>X</td>
<td>Eqn 5.21</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>X</td>
<td>Eqn 5.22</td>
</tr>
<tr>
<td>( M )</td>
<td>50</td>
<td>Eqn 5.25</td>
</tr>
<tr>
<td>( T_{ds} )</td>
<td>0.5s</td>
<td>Eqn 5.26</td>
</tr>
<tr>
<td>( T_{ss} )</td>
<td>0.5s</td>
<td>Eqn 5.30</td>
</tr>
<tr>
<td>( \omega_0 )</td>
<td>( \sqrt{g/z_0} )</td>
<td>Eqn 5.27</td>
</tr>
<tr>
<td>( K_{p1} )</td>
<td>10</td>
<td>Eqn 5.28</td>
</tr>
<tr>
<td>( K_{d1} )</td>
<td>3.3</td>
<td>Eqn 5.28</td>
</tr>
<tr>
<td>( K_{p2} )</td>
<td>50</td>
<td>Eqn 5.28</td>
</tr>
<tr>
<td>( K_{d2} )</td>
<td>16.6</td>
<td>Eqn 5.28</td>
</tr>
<tr>
<td>( A )</td>
<td>-0.1m</td>
<td>Eqn 5.29</td>
</tr>
<tr>
<td>( \tau_1 )</td>
<td>( T_{ds}/2.0 )</td>
<td>Eqn 5.29</td>
</tr>
<tr>
<td>( \tau )</td>
<td>( T_{ss}/2.0 )</td>
<td>Eqn 5.30</td>
</tr>
<tr>
<td>( B )</td>
<td>0.1m</td>
<td>Eqn 5.30</td>
</tr>
</tbody>
</table>

Variables indicated by \( \times \) in Table 5.1 will be defined in next section when we design simulation tests. Most of parameters are like static walker, specially feedback gains. We also have few variables defining swing trajectories. Decreasing CoM height is as before to avoid singularities in knee joints.

### 5.3 Simulation of walking with different speeds

In this section, we want to investigate the effect of \( \text{step}_{max} \), \( N \), \( \alpha \) and middling policy\(^1\). By middling we mean the strategy to choose middle of last two foot

\(^1\)Watch movies at [http://biorob.epfl.ch/page-96274.html](http://biorob.epfl.ch/page-96274.html)
locations in a N-capturable method as final capture point, rather than choosing the last foot location. If middling is chosen, the robot stands normally after N steps and we expect this policy to damp sagittal motions. $step_{max}$ is actually determining the desired speed. We use a ramp function to gradually accelerate forward, defined as:

$$f_{ramp}(A, t, \beta) = \text{Saturate}(At, \beta)$$

Which truncates $At$ when it reaches $\beta$ (at time $t = \beta/A$). For our tests we use $step_{max} = f_{ramp}(0.05, t, 0.3)$ which means reaching $0.3m$ in $6s$. It is desired now to search over all parameters and see which set of them performs better in terms of forward speed and smoothness of trajectories (i.e. lower accelerations needed). Fig[5.9] shows sequences of stepping for various parameter sets.

Plots in Fig[5.9] are compared to choose best parameter set that satisfies our goals. We want the robot to have minimal sagittal motion and accelerations while having maximum speed. Comparing similar graphs which only differ based on middling strategy, one concludes that:

- If $N = 1$, then middling produces sharp CoM trajectory. But if $N = 5$, smoothness is quite similar. Therefore, one would prefer to have larger $N$, meaning future planning.

- Among those graphs with $N = 5$, note that if middling is disabled, trajectories are smoother in the beginning of the sequence, when the robot accelerates forward. Thus one would prefer not to have middling strategy in speed transitions. During steady state motion, middling policy does not make notable difference.

- The choice of $\alpha$ however in this case affects only sagittal motion and one would prefer to minimize it, unless feet do not collide.

Overall we conclude that it is better for capture point to fall on foot places not middle points, and to have large $N$ to plan more future steps.

Let us focus on the plot labeled "F-0.13-5" in Fig[5.9] where middling is disabled, $\alpha = 0.13$ and $N = 5$. One would be interested to compare this figure with Fig[4.12] in previous chapter. Note the trajectory of CoM shown in red lines which goes left and right quite largely there with static walker. But the planner here can predict CoM motion when the robot is in swing phase, enabling us to release the assumption of having a support polygon for controlling CoM. So in Fig[5.9] CoM almost always falls outside polygons (only in swing phases). This produces more natural motion in terms of reducing sagittal motion. Note that $2.5m$ in Fig[4.12] is passed in $50s$ ($0.05m/s$), while $4m$ is passed in $20s$ here ($0.2m/s$) in Fig[5.9].
Figure 5.9: Comparison of different scenarios using various sets of parameters. Each two adjacent plots differ only in one parameter to make the comparison simpler. Note the legends where $T$ means middling strategy is enabled and $F$ means not (i.e. last capture point is last foot step location), 0.07$m$ or 0.13$m$ refer to $\alpha$ and 1 or 5 refer to $N$. 
It worth comparing swing trajectories, as our policies differ in static walk and here. Swing trajectories corresponding to plot labeled ”F-0.13-5” in Fig.5.9 is shown in Fig.5.10.

Figure 5.10: Swing trajectories corresponding to plot labeled ”F-0.13-5” in Fig.5.9.

Comparing Fig.5.10 with Fig.4.12, one concludes that with the new policy of having exponential transitions between old and new positions in Eqn.5.30 the foot lifts off quite faster compared to the simple step strategy smoothed by exponentials in Eqn.4.57. Note that here, we give enough time to the sinusoidal to lift the foot off from the ground before going much forward to the new position. However in Eqn.4.57 we are in fact applying a step function, smoothed by exponentials which move the foot forward quite fast and thus the sinusoidal takes effect mostly at the end of trajectory, resulting in an asymmetric arc.

Since one would be interested in assessing the performance of MPC controller, we visualize instantaneous capture points $P_{ic}(t)$ and their desired points $P_{ic}(0)$ given to MPC in Fig.5.11. Note that there is an intrinsic redundancy in capture point policy which we solve by minimizing acceleration magnitudes to avoid sharp motions.

Main points concluded from Fig.5.11 could be summarized as:

- Although some momentary sharp changes happen in the trajectories of CoM due to hybrid state transitions, the MPC controller smoothly navigates the capture point to the desired point.

- Note that in Fig.5.11, green and black points are calculated by back-propagation of desired conditions in future $N = 5$ steps.

- Due to sliding or bouncing effects and generally any mismatch between controller state or robot state, green and black points are multiple points in fact in a short region. Although there are some small shifts, but they settle
Figure 5.11: Capture point trajectories corresponding to plot labeled "F-0.13-5" in Fig.5.9. These points are plotted only in double support phase so as to show the performance of MPC and smoothness of its response. We also show next desired foot steps $P_{\text{base,new}}$ to measure the performance of feedback gains designed for foot trajectories.

down quickly (compared to $T_{ss}$) to make MPC and foot trajectory controllers, tracking a constant desired point.

- The ideal case was having double support phase durations to be zero. However since at each step, LIP is not ideal as expected, it takes time to adapt new conditions and maintain average speed.

- Based on green dots, one could argue that arc trajectories are not perfectly followed. This tracking could be improved by choosing better feedback gains.

- In case of $N = 1$, green points match with black ones provided that middling policy is disabled.

Swing dynamics affect ideal LIP dynamics because of hip torque in LIP model with mass inertia in Eqn.5.11. Since we do not use large posture correcting gains to improve stability, swing dynamics affects posture of the robot and thus, CoM dynamics. In Fig.5.11, we have chosen $N = 5$ which re-plans next step regarding rest condition in 5 future steps. Since $N$ is relatively small, $R_{ic}(0)$ calculated for this steps will capture more energy from the robot compared to the case of choosing $N = 100$ for example. With later choice, speed variations are smaller in 5 future steps.

Fig.5.11 reveals that swing trajectory tracking is not perfect with current choice of feedback gains. Note that increasing them may sometimes lead to constraint violation in low level controller’s QP problem. Thus one should either consider more ground clearance or larger $T_{ss}$ to improve this effect. These parameters could vary regarding the size of step we want to take. Better tuning could be achieved by optimizations or try-and-error procedures which is out of the scope of this research.
Changing parameters

We would like to know maximum performance of controller in this chapter. With the same configuration of parameters, except $\alpha = 0.07$ and clearance of $B = 0.15 m$ as in Table 5.1, the robot can go up to $4.6 m$ in 20s compared to $4.1 m$ in Fig 5.9, plot labeled "F-0.07-5". As expected, making clearance factor larger would result in better arc trajectory tracking. With the same clearance of $B = 0.15 m$ now, the robot can stably walk with $\beta = 0.35$ in Eqn 5.31, improving total distance traveled to $4.8 m$ in the same timespan. Switching back to $B = 0.1 m$ and $\beta = 0.3$, if we increase $T_{ss}$ in Table 5.1 to 0.6s, the performance again improves by traveling 4.4m compared to 4.1m in Fig 5.9, plot labeled "F-0.07-5". This verifies our claim of improving arc tracking by giving more time.

Rapid speed change

The rate of forward acceleration is determined by the ramp function in Eqn 5.31. With the same set of old parameters "F-0.07-5", but $A = 0.5$ in Eqn 5.31, the robot can step forward generating the sequence shown in Fig 5.12. In this figure, note the first step taken by right foot (bottom sequence) which is far from initial place of foot at rest condition. An advantage in this method is that the robot can accelerate fast by falling. Remember that giving LIP accelerations to the low level controller during single support phase is equivalent to falling which is predicted before by capture point. This extra energy is captured naturally by next foot step. Our method is not using support polygons to apply ankle torques and accelerate forward or decelerate to capture energy.

![Figure 5.12: The output sequence of steps, when the robot is given a desired speed with short time to accelerate forward ($A = 0.5$ in Eqn 5.31).](image)

Push recovery

To wrap up this section, we apply forces to the robot during walking in order to find out how robust it is against external perturbations. In [49], lunging strategy is
used to recover from pushes since the authors assume inertia for the body. However the final body posture after recovery is not the same as beginning which needs another recovery phase and not desirable in humanoid continuous walking.

In contrast, 52 proposes a stepping method which determines a new step location based on online state of the robot. The authors use a learning method to find this point since their robot has distributed mass and it is not easy to calculate capture point analytically. Note that if the robot has foot, the capture point is not just a single point and it includes a region within which the robot can modulate CoP to make CoM resting without taking any step. Finding this region is complicated and requires such heavy learning.

In the controller proposed for this chapter, we do not deal with push recovery in hybrid states. If a push is applied to the robot, it disturbs the predicted state of the system. In single support we do not have any control. However when the robot goes to double support, we recover from the push by controlling over continuous states. Specifically, MPC controller navigates the disturbed instantaneous capture point $P_{ic}(t)$ to the desired $P_{ic}(0)$. The method introduced in next chapter recovers from pushes by taking steps like 53 which uses principles introduced in 37.

To test push recovery in this chapter, the robot walks with our normal set of parameters like plot labeled "F-0.07-5" in Fig.5.9 but we apply pushes to the pelvis in different directions every 5s and let the robot retrieve its normal sequence.

![Figure 5.13: Push recovery scenario for parameter set of plot labeled "F-0.07-5" in Fig.5.9. Here we apply pushes of 20N lasting 0.5s to the pelvis of the robot at times 5s to the left, 10s to the front, 15s to the right and 20s to the back.](image)

Fig.5.13 shows the response of the system to these pushes. Stability highly depends on when we apply the push and to which direction. If the push is in opposite direction of CoM speed, it absorbs energy of the robot and make the control easier. However if it has the same direction, the recovery becomes difficult. In Fig.5.13 all pushes except the one in back direction are in the same direction with CoM velocity, explaining why maximum tolerable push is not very large. The hardest recovery happens around $x = 3m$ after right push. Note CoM trajectory which gets close to support polygon’s boarders.

It is also important to have in mind that the timing of pushes affects control performance severely. In double support or stance phase since MPC controller is
involved, it is easier to recover pushes while in swing phase the robot is falling and we have no control. Note that the timing of double support and single support phases is not as shown in Fig.5.13 with blue and green lines. In fact during double support or stance phase, CoM has higher speeds.

5.4 conclusion

In this section, we proposed a controller which tried to reduce sagittal motion during walking. It was based on simplified model of the robot, i.e. linear inverted pendulum to predict the future motion and to free the assumption of having support polygon in single support. Thus the new controller has intrinsically mechanism to control the robot over hybrid states.

We used the capture point concept introduced in [47] which is a foot placement policy that absorbs all the energy in a LIP. We used this concept for the robot and planned its trajectory, bouncing between predetermined sequence of future steps where we assumed rest condition at the end. By back-propagating, we calculated the desired capture point for a robot in double support phase and we used an MPC controller to reach that point. Overall, we can enumerate advantages of this method as:

- **Exact foot placement**: In our method, we were basically determining the exact position of next foot step in the sequence of $N$ future steps. Although arc tracking in swing phase could have been improved by better feedback tuning, but the robot was still able to maintain the expected sequence as MPC was re-planning the capture point.

- **Symmetric sequence**: Recall from static walking in previous chapter that at each step, we were giving a fixed shift to each foot individually, having the risk of accumulating errors. Here however, when we plan $N$ future steps, we preserve symmetry of subsequent steps by adding shifts with respect to the position of other foot.

- **Smoothness**: Is achieved using the MPC controller which predicts the future path of CoM in stance phase. Note that a PD controller might have overshoots or other delays while MPC benefits from system model and reaches desired point smoothly.

- **Redundancy**: Since capture point is a combination of position and velocity, there are multiple solutions and MPC has the opportunity to reject possible disturbances by changing the solution within the null-space of final desired point.
• **Higher speed**: Compared to static walking in previous chapter, the performance of this controller is much better, achieving $0.2 \text{m/s}$ while there, maximum speed of static walking was around $0.06 \text{m/s}$. Note that the similar capture point approach taken by [51] reaches $0.1 \text{m/s}$ on average. This is the nature of ZMP control used in their low level controller.

Despite having all these advantages, the method has important drawbacks which make us thinking of a more advanced controller in next chapter:

• **Double support**: Usually in human walking, the assumption of full foot contact is not true because of toes. Thus one can argue that double support has smaller time compared to single support in human walking. This motivates us to move disturbance correction to hybrid states and improve walking speed. The method in this chapter needs double support phase to correct disturbances over continuous states.

• **Disturbance**: As we deal with them in double support phase, the method relies on having support polygon which is not desired. Although in many humanoid robots, feet are quite large, but in Atlas they are rather small and we have limited control.

• **Redundancy**: Although redundancy has positive points, but we must solve it either having assumptions on speed and position or minimizing accelerations which is used here. One is therefore interested in using this computational power somewhere else.

Although exact foot placement seems to be an advantage for this method, but considering LIP model and capture points, one can simply absorb energy of the robot over hybrid states by slightly changing next foot locations. Note that humans also can have at most one pre-determined future step and after that, they try to balance by taking different steps. Moreover when they are required to take many fixed steps, they use double support phase consciously. In next chapter, we will move the optimization to hybrid states rather than capture points and remove stance or double support phase from controller state machine.
Chapter 6

Control Method 2: Dynamic walking

In previous chapters, we proposed two different walking controllers relying on support polygon. The first one was static controller using polygon both in double support and single support. It was always keeping CoM inside the polygon, having full control over the robot. The second one however released the assumption of having support polygon in single support, but still needing it during double support to compensate for unwanted effects. In this chapter, we want to release the second assumption as well and perform dynamic walking, i.e. not using support polygons.

Using support polygon is not reliable, specially in case of Atlas robot whose feet are quite small. We also want to avoid rolling effects of foot on the ground, assuming that it is always in full contact. In fact fast walkings could not be achieved with assumption of full contact, since it necessitates taking large steps and ankle joints which have pivots along y direction (“l-leg-uay” and ”r-leg-uay” in Fig.3.1) have angle limitations which restrict foot steps sizes. We also want to avoid singularities by decreasing height of pelvis which then again limits foot step sizes. In this research, we assume full contact and try to find walking principles with maximum speed possible and desired stability. Using toes and rolling of foot for faster walking is our future work which needs more advanced contact modeling.

As mentioned, we release the assumption of having support polygon in all phases. Thus we assume that the robot has point contact. Remember the argument we had in previous chapter on inertia of the robot’s body and hip torques correcting posture errors. These torques often causes CoP not to coincide with projection of ankle on the ground. In fact we give LIP accelerations and assume that CoP coincides with projection of ankle provided that the robot do not take large steps. Recall that not necessarily, reference hybrid state (imposed by controller) should match actual hybrid state (in simulation). The controller might impose left support for example and generating zero contact forces at the right foot, while in reality
the right foot has still not lifted off. These mismatches have negligible time, but important to consider.

To reach higher speeds, we remove double support phase from the controller and directly switch from left support to right support and vice versa. The low level controller takes care of actuator torque rate limitations and thus bouncing effects may not happen. By assuming no support polygon we do not have ankle torques, so double support phase does not make sense. The robot is therefore always in single support phase and we can calculate CoM accelerations based on LIP model approximately. The only control we have on the robot is therefore where to put its swinging foot. In previous chapter, on top of the low level controller we had a second QP problem, formulated in the form of an MPC controller which had to navigate current capture point to the desired point during single support phase. This optimization was done in planner level (See Fig.4.1) and on continuous time variables. However in this chapter we would like to move it to navigator level and make planner very simple. The navigator thus gives planner only a Cartesian point as next desired location to place the foot. In fact planner controls continuous variables while navigator controls hybrid states.

Another problem we would like to solve is intrinsic redundancy in capture points. Recall that LIP model has a linear differential equation which is solvable in closed form. Therefore we can analytically find the approximate future trajectory of CoM when falling. Such prediction will help the robot recover from larger pushes as well. Having briefly described our method, we also review some important works inspiring the method we will propose in this chapter.

6.1 Literature review on planning methods

In [54], authors propose a method similar to capture points called Foot Point Indicator. They assume a point-foot robot with arbitrary number of non-massless links. The method finds a capture point where if the robot steps, the total energy in the system is captured. Assuming zero energies after impact, the method finds a series of desired joint angles before impact which PD controllers have to follow. To compare, this method has similar nature of point feet as we have assumed implicitly. However not only it considers 1-Step capturability, but also the basis of this method is position controlled joints which is not desired in our case as we need compliance. Compared to 1-Step capturability policy used in [47], this method is more complete in the sense that it does not simplify the robot.

In [55], a hierarchical approach is proposed which is perceived principally, i.e. it can detect height changes in the terrain and adapt to them. In low level controller, a QP optimization problem generates joint torques knowing Cartesian end-point forces and positions. Given a goal location for CoM and feet, a path planner planes
half-cycle trajectories in Cartesian space. These trajectories are optimized off-line to minimize error. They are tracked online by controllers generating Cartesian forces which are then given to low level controller. The path planner adapts trajectories if terrain changes. On top of all, a locomotion controller determines goal locations per half-cycle. To compare, our low level controller is principally compliant since it uses dynamics model, not the kinematic model used here. We also simplify robot’s model and plan Cartesian paths on-line which is more robust and general. This work however has steering properties which we do not address in our project.

In [45] as we described before, the low level controller is similar to ours. Whitman simplifies the problem into Cartesian accelerations like [44] and he uses dynamic programming to obtain policies for walking control. He in fact decouples Cartesian variables and optimizes for periodic policies off-line which take very long time, but not affecting online performance. Our method rather finds policies online using simplified model of the robot which is not used in [45].

In [56], the low level controller is the same as what we use here, but planning method is different slightly. This work models the robot with Spring Loaded Inverted Pendulum (SLIP) in stance phase which enables CoM’s height to vary during walking. It also models robot with a projectile during flight phase. With some simplifying assumptions, the authors can express stance and flight trajectories by polynomials in closed form. Thus considering a specific sequence of single-support / double-support / flight phases, they can produce specific motions such as walking or running. Various criteria are used to make the walking natural, periodic, robust and fast. So they have a planning optimization which optimizes future steps and finds optimal control policy and Cartesian trajectories. The method can also be perceived which enables it to have exact foot-placement. To compare, since they use a more complicated model, their optimization takes considerable time and has many local minima. If they plan per time-step, they achieve real time factor of only 5% as they claim. We inspire this idea in this work and design our own planner similarly. One can use simpler model of the robot and relate states over hybrid phases linearly together. We will show this by using LIP instead of SLIP. With such assumption, we can optimize states in the future more easily and per time-step. In our future works, we try to include flight phases and height changes as well.

In [53], similar platform is proposed based on theoretical background developed in [47]. The idea of capture points is applied on a 3D force controlled robot which has two tasks of maintaining balance and walking. The low level controller in this method is similar to ours, in the sense that it optimizes for joint torques without needing any trajectory like [51] for example. The method proposed in this work calculates a capture region where if the robot steps inside, it can go to
rest condition without taking any further step. They can adjust this region with respect to current capture point of the robot which enables them to respond to pushes online.

To compare [53] with our work, they simplify their robot model to LIP with foot whereas we simplify it with LIP itself, thus having a capture point instead of capture region. With such simplicity we can plan more than one step into the future in this chapter while [53] can only plan 1 step. They also rely on the assumption of having foot and ankle torques while we do not rely on these torques and have a completely dynamic walking. However our push recovery with stepping strategy and compliant low level control is similar to their work.

6.2 Model predictive control: Step planning

An advantage the controller we had in chapter 5 was in fact the ability to plan for exact foot placement. One can impose the next desired foot location when calculating the fixed sequence of steps in navigator of previous chapter (Eqn [5.21]). Back propagation will find the desired capture point $P_c(0)$ which is then given to MPC. Controlling this capture point is however depending on having polygon of support which is not desired. In this chapter, we propose a navigator that plans future steps automatically. Although one can restrict the first foot place, but the robot needs enough freedom to plan next ones. This is also true if the robot wants its $N^{th}$ step being at some desired location. For the purpose of this research, we aim at designing simple navigator and investigate these options in future works.

Fig. 6.1 shows the interconnection of different blocks. We determine next desired foot place $P_{base,new}$ in navigator and planner has to follow it. The planner is similar to the one proposed in previous chapter except that it does not have to follow any capture point, since there is no double support phase. We purely control the robot with hybrid dynamics and let LIPM determine accelerations inside the planner.

![Figure 6.1: Combination of different blocks used to control the robot in this chapter.](image)

6.2.1 Navigator design

The navigator we design here gets current states of CoM (position and velocity) and plans next foot step, minimizing some objective function. Before explaining

99
this optimization, we would like to solve LIPM equations analytically.

**Solving LIPM differential equation**

In this part we solve Eqn.5.5 analytically to find exact evolution of CoM trajectory over hybrid steps. Specially we would like to know what would be the state of robot after swing time and right before touch down, given CoM state in the beginning of that swing phase. Note that these equations are true when there is no hip torque. So they approximate our robot’s behavior. We re-state Eqn.5.6 again:

\[
\ddot{x} = \frac{g}{z_0} (x - x_{base}) \\
\dot{y} = \frac{g}{z_0} (y - y_{base})
\]  

(6.1)

We solve the first equation and the solution is therefore similar for the second one. One can re-write this equation as:

\[
\ddot{x} = ax + b
\]

(6.2)

where:

\[
a = \frac{g}{z_0} \\
b = -\frac{gx_{base}}{z_0}
\]

(6.3)

Taking the Laplace transform from both sides, we have:

\[
X(s) = \frac{b}{s^2 - a} = \frac{b}{2\sqrt{a}} + \frac{-b}{2\sqrt{a}} \left( \frac{s}{s - \sqrt{a}} + \frac{s}{s + \sqrt{a}} \right)
\]

(6.4)

A Laplace transform with this form could be written in time domain as:

\[
x(t) = Ae^{-t/\tau} + Be^{t/\tau} + C
\]

(6.5)

By examining \(x(0)\) and \(\dot{x}(0)\) in this equation we can obtain:

\[
\tau = \sqrt{\frac{g}{z_0}} \\
C = x_{base} \\
A = \frac{-\tau \dot{x}(0) + x(0) - x_{base}}{2} \\
B = \frac{\tau \dot{x}(0) + x(0) - x_{base}}{2}
\]

100
One can then calculate $x(T)$ in terms of $x(0)$ and $\dot{x}(0)$ by:

$$x(T) = Ae^{-T/\tau} + Be^{T/\tau} + C$$

(6.7)

$$x(T) = x(0)(\frac{e^{-T/\tau}}{2} + \frac{e^{T/\tau}}{2}) + \dot{x}(0)(\frac{-\tau e^{-T/\tau}}{2} + \frac{\tau e^{T/\tau}}{2}) + x_{base}(\frac{-e^{-T/\tau}}{2} + \frac{-e^{T/\tau}}{2} + 1)$$

One can also calculate derivative of Eqn. 6.5 analytically and find $\dot{x}(T)$ in terms of initial conditions. To simplify, if we define $\omega_0 = \frac{1}{\tau}$ and $h = e^{T/\tau}$, we will have:

$$\begin{bmatrix} x(T) \\ \dot{x}(T) \end{bmatrix} = \begin{bmatrix} \frac{1}{2\tau h} + \frac{h}{2} & -\frac{1}{2\omega_0} + \frac{h}{2} \\ -\frac{1}{2\omega_0} + \frac{1}{2} & \frac{1}{2\tau} \end{bmatrix} \begin{bmatrix} x(0) \\ \dot{x}(0) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} x_{base}$$

(6.8)

We can therefore symbolize the following equation by:

$$\begin{bmatrix} x(T) \\ \dot{x}(T) \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} x(0) \\ \dot{x}(0) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} x_{base}$$

(6.9)

Since constant matrices are only functions of time $T$ and constant $\omega_0$, one can include $y$ variable in Eqn. 5.6 as well and relate full states of CoM at time 0 and $T$ as:

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} (T) = \begin{bmatrix} a_1 & 0 & a_2 & 0 \\ 0 & a_1 & 0 & a_2 \\ a_3 & 0 & a_4 & 0 \\ 0 & a_3 & 0 & a_4 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} (0) + \begin{bmatrix} b_1 \\ 0 \\ b_2 \\ 0 \end{bmatrix} \begin{bmatrix} x_{base} \\ y_{base} \end{bmatrix}$$

(6.10)

Which could be symbolized again as:

$$q(T) = A(T)q(0) + B(T)P_{base}$$

(6.11)

Where $q$ is the full state of the system. Note that this equation in fact relates the full state of the system at the beginning of swing phase to the end. Therefore, assuming no double support, we have the possibility of predicting state of the system after one step, provided that we know step locations $P_{base}$.

**N-Step foot placement in future**

As obtained in previous section, we now have the possibility of predicting future evolution of CoM trajectory. Remember that at each instant of time, the navigator reads CoM position and speeds and plans for future. We also assume that swing
phases all have the same duration $T$. We can then define the following sequence of states and step locations as:

\[ q[1] = A_0 q[0] + B_0 P[0] \]
\[ \vdots \]
\[ q[N+1] = A q[N] + B P[N] \]

Where $A_0 = A(T - (t - t_0))$, $B_0 = B(T - (t - t_0))$ and $t_0$ is the beginning time of current swing phase. They are different because when the robot is in swing phase started at $t_0$, already $t - t_0$ is passed from start time of swing and $T - (t - t_0)$ is remained nominally. In Eqn. 6.12 we know $q[0]$, $A_0$, $B_0$, $A$, $B$ and $P[0]$ and we would like to know $P[1]$, the next place swing foot should go.

**Controllability**

Eqn. 6.12 is in fact a sequence with states and control inputs which could be thought of as a discrete dynamical system, with model matrix $A$ and input matrix $B$. Although $A_0$ and $B_0$ are different, but note that the result of first equation $q[1]$ is already available and thus, one could imagine the system start from $q[1]$. From modern control theory, we know that for a discrete system described with:

\[ x[k+1] = Ax[k] + Bu[k] \] (6.13)

Where $A \in \mathbb{R}^4$ and $B \in \mathbb{R}^4$, controllability could be checked by examining the rank of $C$ matrix defined by:

\[ C = [B \ AB \ A^2B \ A^3B] \] (6.14)

With parameters $g = 9.8$, $z_0 = 0.84$ and $T = 0.5$, the matrix $C$ has full row rank of 4, meaning that the system is controllable. Now for general case, if we prove that the matrix $D$ defined by:

\[ D = [B \ AB] \] (6.15)

has full rank, then we can conclude that $C$ is also full row rank and the system is controllable. Using symbolic functions of MATLAB, one obtains:

\[ \det(D) = (-w_0 h^4 + 2w_0 h^3 - 2w_0 h + w_0)^2/(4h^4) \] (6.16)
\[ \det(D) = (-h^4 + 2h^3 - 2h + 1)^2w_0^2/(4h^4) \]
\[ \det(D) = (1 - h)^6(1 + h)^2w_0^2/(4h^4) \]
It is obvious that $\text{det}(D)$ is always positive and becomes zero only if $h = 1$. Remember that $h = e^{T/\tau}$ and:

$$h = 1 \iff T = \tau = \sqrt{\frac{g}{z_0}} \approx 3.41 \text{s} \quad (6.17)$$

This means that swing duration should take $3.41 \text{s}$ which is far more than our choice of $T = 0.5$. So we can make sure that around time constant of the system, our choice of $T$ does not make the system uncontrollable.

**Objective, constraints**

Knowing that the system is controllable, by definition we can conclude that it can go from $q[0]$ to any $q[N]$ in a limited time $N$. Although there are various modern control methods to design controllers for this system like state feedback and etc., we would like to solve the problem in a discrete-time MPC formulation where we minimize least square errors of future states with a given desired average speed. This is in fact tracking problem which does not consider position, but only speed. Note that sometimes we want the robot to walk sideways and sometimes forward or backward.

Remember from previous chapters that placing feet closer (along $y$ direction) will reduce sagittal motion. So if we formulate our tracking MPC with least square errors of speeds, feet are placed close and close and finally they coincide which should be avoided due to self-collision. This happens because the MPC wants to minimize sagittal motion, given zero desired sagittal average speed. We thus need to add another mechanism penalizing this effect. One can introduce constraints in the optimization, for example left foot should always be at the left side of right one and vice versa. This specific hard constraint does not work specially for large sagittal pushes. One can consider the shape and solve self collision for feet within optimization. However this constraint is not convex and thus not easy to solve. Note that we also have the constraints of $step_{\text{max}}$ similar to Eqn.5.21 which could be added here. We can thus control maximum length of steps.

To avoid feet getting close to each other, we penalize them in the objective function. To this end, remember Eqn.5.21 where a series of foot steps were generated. Similarly we can plan two subsequent steps where the latter should be ideally the former plus some constant shift. We can then calculate a summation of least square errors and add it to the objective function.

**Quadratic Problem (QP) formulation**

So far, all objectives and constraints are introduced. We have equations describing the state of robot over $N$ future steps and we want to find the best place to put
swinging foot. This optimization is done per time step based on current state of CoM, which enables the robot to recover from pushes over hybrid states. Our control input to the planner is therefore next desired foot place. We can formulate such navigator optimizer as:

\[
\begin{align*}
q[1] &\in \mathbb{R}^4, \quad P[0] \in \mathbb{R}^2, \quad A \in \mathbb{R}^{4\times4}, \quad B \in \mathbb{R}^{4\times2} \\
s_1 \in \mathbb{R}^2, \quad s_2 \in \mathbb{R}^2, \quad v_{\text{ref}} \in \mathbb{R}^2, \quad N \in \mathbb{N}
\end{align*}
\]

\[
\min_{P[k], q[k]} \sum_{k=2}^{k=N+1} \quad \text{quad}(q[k][3:4] - v_{\text{ref}}) + \\
\sum_{k=1}^{k=\text{floor}(\frac{N}{2})} \quad \text{quad}(P[2k-1] + s_2 - P[2k]) + \\
\sum_{k=0}^{k=\text{floor}(\frac{N-1}{2})} \quad \text{quad}(P[2k] + s_1 - P[2k+1]) \\
\text{s.t.}
\begin{align*}
q[k+1] &= Aq[k] + BP[k] & k = 1 : N
\end{align*}
\]

In this procedure, inputs are:

- \(N\): number of future steps.
- \(q[1]\): state of the system right at next touch down.
- \(P[0]\): position of contacting foot (\(P_{\text{base}}\)).
- \(A\) and \(B\): constant matrices in Eqn.6.10.
- \(s_1\) and \(s_2\): shift vectors used to create ideal foot sequence (described later).
- \(v_{\text{ref}}\): desired speed of the robot.

And we get \(P[1]\) as next desired foot position. In fact, we minimize least square error of CoM speed with desired speed and also least square error of foot positions with ideal positions. \(s_1\) and \(s_2\) are defined as:

\[
\begin{align*}
s_1 &= \begin{cases}
[T v_{\text{ref}}.x & -d & 0] & \text{left support} \\
[T v_{\text{ref}}.x & d & 0] & \text{right support}
\end{cases} \\
s_2 &= \begin{cases}
[T v_{\text{ref}}.x & d & 0] & \text{left support} \\
[T v_{\text{ref}}.x & -d & 0] & \text{right support}
\end{cases}
\end{align*}
\]
Variables \( s_1 \) and \( s_2 \) are defined as shifts between subsequent foot-steps. The green colored swinging foot has to touch down at \( P[1] \), which is the outcome of QP optimization.

Where \( T \) is swing duration used in Eqn. 6.7 and \( d \) is ideal distance between feet. Fig 6.2 shows such ideal sequence given \( P[0] \). The green colored swinging foot has to touch down at \( P[1] \).

Since we used current state of the system (to calculate \( q[1] \) by Eqn. 6.10 which is given to the optimization), we have the ability to respond to perturbations or external pushes on-line. The optimization is therefore responsible to adapt \( P_{\text{base,new}} = P[1] \) in order to capture extra energy in the system. Note also that we build future step sequence incrementally. Although one can calculate this sequence only based on \( P[0] \), our formulation is more flexible since implicitly, more emphasis is considered for tracking desired velocity \( v_{\text{ref}} \) in the objective function. More precisely, with incremental formulation, the linear cost matrix \( R \) in a standard QP formulation (like Eqn. 4.28) will have mostly zero elements. Linear cost penalizes optimization variables when they have small values while quadratic cost penalizes when they have large values. No linear cost means that foot places \( P[k] \) could have larger errors in the final solution, causing speeds have smaller errors and be more optimized.

Note also that if we do not use an ideal sequence, minimizing sagittal motions (in the direction of \( y \) axis) will cause both feet coincide finally. We penalize for such effect by using variable \( d \) to avoid self collisions. We also shift subsequent steps forward (in the direction of \( x \) axis) based on desired velocity \( v_{\text{ref}} \) to form ideal sequence. So far the navigator block is designed which determines next desired foot position. We will now, describe our simple planner block who follows trajectories and generates Cartesian accelerations, to be given to the low level controller (see Fig. 4.1).

### 6.2.2 Planner design

As described before, in this controller we do not have double support phase. Thus we only have two phases (compared to three in previous chapter) and LIP model during single support phases to generate CoM accelerations. Note that in all
scenarios the robot starts from stance phase. Here we have stance phase only for 1s during which we apply an acceleration of 0.13m/s² to the left. So the robot gets prepared for starting dynamic phases. This is actually a better initial condition for the system which reduces initial stumbling.

We have shown state machine used in planner block of this controller in Fig. 6.3. This state machine is similar to the one in previous chapter Fig. 5.8 except that it does not have double support phase in the loop. The swing function is again defined as:

\[
f(t, P_{prev}, P_{new}) = \begin{bmatrix} 
    P_{prev,x}(e^{-\frac{(t-t_0)^2}{\tau^2}}) + P_{new,x}(1 - e^{-\frac{(t-t_0)^2}{\tau^2}}) \\
    P_{prev,y}(e^{-\frac{(t-t_0)^2}{\tau^2}}) + P_{new,y}(1 - e^{-\frac{(t-t_0)^2}{\tau^2}}) \\
    B\sin(\frac{\pi (t-t_0)}{T}) 
\end{bmatrix}
\]  

(6.20)

Where \( P_{new} = P_{new,base} \) is the next desired foot position. This planner has the capability to adapt to \( P_{new,base} \) if it was changes by navigator in response to pushes for example. Such effect is seen in simulation tests later. We generate accelerations
to be given to the low level controller by:

\[
\begin{bmatrix}
\omega_0^2 (P_{CoM,x} - P_{base,x}) \\
\omega_0^2 (P_{CoM,y} - P_{base,y}) \\
K_p (P_z - P_{CoM,z}) + K_d (\dot{P}_z - \dot{P}_{CoM,z}) + \ddot{P}_z
\end{bmatrix}
\] (6.21)

\[
\ddot{P}_{\text{left}} = \begin{cases}
K_p (P_{L,des} - P_L) + K_d (\dot{P}_{L,des} - \dot{P}_L) + \ddot{P}_{L,des} & \text{if right support} \\
0 & \text{else}
\end{cases}
\]

\[
\ddot{P}_{\text{right}} = \begin{cases}
K_p (P_{R,des} - P_R) + K_d (\dot{P}_{R,des} - \dot{P}_R) + \ddot{P}_{R,des} & \text{if left support} \\
0 & \text{else}
\end{cases}
\]

Where \(\omega_0 = \sqrt{\frac{g}{z_0}}\) and \(P_{base} = P[0]\) used in navigator. CoM height is controlled by:

\[
P_z = 0.94 + Ae^{-\frac{t^2}{\tau^2}}
\] (6.22)

And Cartesian foot trajectories are:

\[
P_{R,des} = f(t, P_{R,f}, P_{base,new})
\]

\[
P_{L,des} = f(t, P_{L,f}, P_{base,new})
\] (6.23)

Where \(P_{base,new} = P[1]\), coming from navigator. The planner in this chapter is much similar to previous chapter \(5\) except that there is no double support and MPC control.

### 6.2.3 Parameter tuning

There are some parameters to be tuned in the discrete-time MPC controller we proposed in this chapter. These variables are mostly concerning arc trajectories and PD controllers, listed in Table 6.1.

Variables indicated by X in Table 6.1 will be defined in next section when we design tests. As we can see, most of parameters are tuned as before including feedback gains. We also have some variables determining swing trajectories. Decreasing CoM height is like before to avoid singularities in knee joints.

### 6.3 Simulation of walking

In this section, we would test our algorithm for different conditions\(^1\). We want to test highest speed it can reach, speed transitions, pushes and model errors. At

\(^1\)Watch movies at [http://biorob.epfl.ch/page-96274.html](http://biorob.epfl.ch/page-96274.html)
Table 6.1: Values of the parameters used in discrete-time MPC controller.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>X</td>
<td>Eqn.6.18</td>
</tr>
<tr>
<td>$v_{\text{ref}}$</td>
<td>X</td>
<td>Eqn.6.18</td>
</tr>
<tr>
<td>$d$</td>
<td>X</td>
<td>Eqn.6.19</td>
</tr>
<tr>
<td>$T$</td>
<td>0.5s</td>
<td>Eqn.6.10</td>
</tr>
<tr>
<td>$\omega_0$</td>
<td>$\sqrt{g/z_0}$</td>
<td>Eqn.6.21</td>
</tr>
<tr>
<td>$K_{p1}$</td>
<td>10</td>
<td>Eqn.5.28</td>
</tr>
<tr>
<td>$K_{d1}$</td>
<td>3.3</td>
<td>Eqn.5.28</td>
</tr>
<tr>
<td>$K_{p2}$</td>
<td>50</td>
<td>Eqn.5.28</td>
</tr>
<tr>
<td>$K_{d2}$</td>
<td>16.6</td>
<td>Eqn.5.28</td>
</tr>
<tr>
<td>$A$</td>
<td>-0.1m</td>
<td>Eqn.6.22</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>0.5s</td>
<td>Eqn.6.22</td>
</tr>
<tr>
<td>$\tau$</td>
<td>$T/2.0$</td>
<td>Eqn.6.20</td>
</tr>
<tr>
<td>$B$</td>
<td>0.2m</td>
<td>Eqn.6.20</td>
</tr>
</tbody>
</table>

In the end, we will also perform walking on rough terrain and slopes. Remember from previous section that at the beginning of simulation, the robot is standing and we will switch to dynamic walking mode where there is no double support or standing. This procedure takes 1s during which the robot is accelerating to the left to provide better initial condition. We consider a settling time during which the robot starts dynamic walking with zero desired speed and converges to a limit cycle. Note that we should determine $v_{\text{ref}}$ for the navigator. If we give $v_{\text{ref}} = 0$, the robot will walk in place, but with the risk of accumulating errors. So it might go to different directions slowly. In order to keep the robot on straight line, we use a PI feedback defined as:

$$
v_{\text{ref}} = \left[ v(t) + (v(t) - \dot{P}_{\text{CoM},x}) \times 0.5 \right. - \left. \dot{P}_{\text{CoM},y} \times 1.0 - P_{\text{CoM},y} \times 0.5 \right]
$$

(6.24)

Where $v(t)$ is defined as a delayed ramp function:

$$
v(t) = \begin{cases} 
0 & t < t_{\text{settle}} \\
 f_{\text{ramp}}(A, t - t_{\text{settle}}, \beta) & t > t_{\text{settle}} 
\end{cases}
$$

(6.25)

This mechanism is in fact similar to a virtual component connected to the robot like what we explained in chapter 2. Parameters are tuned such that they reduce sagittal motion and provide good tracking of desired speed. Later we will see that these feedback gains are very small. Choosing a normal speed of $f_{\text{ramp}}(0.025, t - 8, 0.3)$ will result in the response shown in fig.6.4. Note that in the beginning ($t < 8s$), few steps are taken to stop stumbling and then the system
speeds up. In Eqn[6.24] we do not add any term corresponding to control of position in $x$ direction. We only control speed that might suffer from incremental error problem.

Figure 6.4: Swing trajectories corresponding to $f_{\text{ramp}}(0.025, t - 8, 0.3)$.

In Fig.6.4, arc trajectories are larger compared to Fig.5.10. We have deliberately chosen $B = 0.2m$ to be able to traverse over rough terrain, avoiding the risk of hitting the terrain in the middle of swing phase. Larger clearances produce unnatural walking and also consume more energy. To analyze CoM trajectory, we plot the same sequence from top view in Fig.6.5

Figure 6.5: CoM trajectory and foot steps corresponding to $f_{\text{ramp}}(0.025, t - 8, 0.3)$.

One can observe the effect of PI controller proposed in Eqn.6.24 on foot step sequence shown in Fig.6.5. In this figure the CoM trajectory has less sagittal motion compared to Fig.5.11 and Fig.5.9 in previous chapter, which is due to eliminating double support phase and making the walking completely dynamic. Performance of PI controller on top of the navigator determines how responsive the robot is. We have shown speed tracking performance for $x$ and $y$ and position tracking for $z$ component of CoM in Fig.6.6

Important points observed in Fig.6.6 could be listed below where bold variable show the corresponding plot:

- $\dot{P}_{\text{CoM},x}$: During settling time, although the robot takes quite large steps in the beginning, but it is able to recover and go to a stable limit cycle. This
is in fact done by MPC controller which plans future steps to capture extra energy in the robot and track desired speed which is zero here. The same effect is observed in $\dot{P}_{CoM,y}$ plot. Note that the navigator is given $v_{ref} = 0$ during settling time.

- $\dot{P}_{CoM,x}$: The average speed in settling time is not zero and the robot goes forward very slowly. This shows incremental error and the fact that we only have feedback over velocity, not the position along $x$ direction. Recall that due to having hip torque in the real robot in contrast to our LIP model, one would expect such effect because of biased model errors.

- $\dot{P}_{CoM,y}$: Speed tracking is better when desired speed is not zero.

- $\dot{P}_{CoM,y}$: When time passes on, although steps in $x$ direction become larger, sagittal motion (along $y$) is the same, meaning that decoupling of the motion into $x$ and $y$ components is acceptable.

- $P_{CoM,z}$: This plot shows that the CoM height controller has considerable steady state variations (up to $2 - 3cm$). Since we do not have double support phase in this controller, there are rather sharp transitions between foot steps. This is because one leg is tolerating the weight of robot and this role is rapidly
transferred to the other leg. So large impacts are appearing which make \( z \) component of CoM fluctuating.

It is also interesting to plot \( \dot{P}_{CoM,x} \) vs \( \dot{P}_{CoM,y} \) when the robot is reaching \( v_{ref,x} = 0.3 m/s \). This plot is shown for 9 limit cycles in Fig. 6.7. One can see a full limit cycle in the middle of these plots where the graph is closed, however the robot has not yet converged to a single cycle. There are many reasons for this behavior such as sensor noises, hybrid states, disturbances, controller-robot communication delay and etc. By definition, our navigator quadratic problem has a positive semi-definite \( Q \) matrix when written in standard form of Eqn. 4.28. Such matrix reduces energy in the system. We also proved controllability of this discrete dynamical system. However due to various factors such as delays and impacts in simulation, we can’t prove stability for limit cycles. Rather we simulate them for a long time (150s) to see if the robot fails or not.

![Graph showing limit cycles of walking motion](image)

Figure 6.7: Limit cycles of walking motion when the robot has reached \( v_{ref} = 0.3 m/s \). The period of walking is about 1s and duration of settling is about 10s. These 9 limit cycles show that the motion does not completely converged to a single cycle. Note that we apply median filter to remove impacts from signal of Fig. 6.6 and mean filter to smooth it.

Finally we show the sequence of footsteps planned by navigator. At each time-step, the navigator is given current state of the robot and it generates a series of foot steps according to an objective function minimizing least square error of this sequence with ideal case. Regarding duration of single support phase which is
$T = 0.5s$, we plot the planned sequence every 0.1s to show its behavior. Fig. 6.8 shows 6 snapshots of this sequence where the robot is in left support phase in the beginning. When it switches to right support, the full blue colored rectangle shows the next desired position of foot $P[1]$ where it should touch down. This point is then given to planner block to generate arc trajectories.

Note that bending in the sequence to left and right is because of PI feedback terms on top of the navigator. So $v_{ref,y}$ is not always zero and the sequence is therefore adapting to this speed. Note that the bending direction is opposite to CoM speed most of the time, because of PI feedback’s correcting behavior. One can add terms penalizing $y$ position of CoM to the objective function of optimization. This in fact makes the sequence straight, while severely affecting $P[1]$, the next step to be taken. This is not desired since $P[1]$ sometimes goes very far from base foot to capture most of the energy and this leads to instability. Other options to have $v_{ref,y}$ nearly zero is either filtering CoM velocity, or giving average speeds over previous steps to the PI controller of Eqn. 6.24 which causes delays and thus instability. The choice of $N$ does not affect the sequence very much. Choosing larger $N$ will produce the same repeated motion. Smaller $N$ also causes sequence capture energy very fast which produces large steps.

Figure 6.8: Sequence of steps planned by navigator during a single support phase. The figure shows a snapshot from this sequence every 0.1s where the duration of single support is $T = 0.5s$. Note the upper left picture where the robot is in left support, about to switch contacting foot (exceptionally blue and red colors are reverse in this plot, denoting stance and swing foot future locations). In the rest of pictures, the swing foot (green) is trying to reach the blue location $P[1]$ gradually.

So far, we discussed different properties of our controller on a the same simulation scenario with desired forward speed of $0.3m/s$. In next parts, we will
investigate other aspects of this controller, including speed variations, robustness to perturbations, model errors, terrain roughness and etc.

6.3.1 Speed variations

In this part, we would like to investigate the range of speeds that could be given to the robot. In all scenarios we assume the same ramp function \( f_{\text{ramp}}(0.025, t - 8, \beta) \) where we change \( \beta \) from 0.0 m/s to 0.5 m/s. Fig 6.9 shows result of different tests in steady state where the robot stabilizes.

One can conclude from Fig 6.9 that:

- The robot is capable of adapting to various desired speeds. This is compared to off-line optimization based methods which could only change speed around the nominal solution.

- Variations in response of the system is natural because of bouncing effect. The robot is in fact always falling while subsequent steps change its direction. Note that CoM accelerates when getting far from stance foot and decelerates when going towards front foot.

- Higher variations are expected when desired speed is larger. However steady state errors are larger when desired speed is smaller.

The rhythm of movement is changing during time because of various unwanted factors, but still being controlled. Having desired speed of 0.3 m/s, with current ramp function \( f_{\text{ramp}}(0.025, t - 8, 0.3) \), it takes 12 s to reach this speed. At minimum, this time could be 3 s which means 3 periods or 6 steps forward, denoting acceleration properties of the robot. Finally One is interested also to know if the robot is able to walk backward or not. In fact with the current tuning of parameters, the robot is only able to go back with an average speed of \(-0.1\) m/s where there is quite large steady state error in tracking of desired speed \((v_{\text{ref}} = -0.2\) m/s). This limitation is mostly because of parameter tuning of PI controller which could be improved. However choosing larger gains may sometimes produce large steps to compensate for tracking error.

6.3.2 External pushes

In this part we investigate the response and robustness of our walking method to external pushes. Such scenario is shown in Fig 6.10 for \( v_{\text{ref}} = 0 \) and pushes being applied to the pelvis.

Ideally, the robot should be able to recover stronger pushes while not having high speeds. We first test this case to see how strong a recoverable push could be
Figure 6.9: Steady state response of the system to different desired speeds. The top plot shows 3 steps and bottom plot shows time-response. Note that we have omitted impacts by median filter and smoothed the graphs by mean filter.
Figure 6.10: Walking locomotion with $v_{ref} = 0$. We apply pushes of 40N each lasting 0.5s to the robot every 5s in different directions. The robot recovers these pushes by coming back to its original place. Note that we have position control over $y$ while there is no control over $x$. The bottom plot shows foot $y$ locations where we have overlap or collision when the robot is pushed to the left.
in each direction. A sample demonstration is shown in Fig.6.10. In this scenario, pushes of 40N are applied to the robot each lasting 0.5s. Note that we had push recovery once in chapter 4 where the robot was in standing mode and able to reorient to keep center of mass inside polygon. We were able to recover pushes of 50N to 75N in different directions. We also tried push recovery in previous chapter where the robot was able to withstand small pushes of 20N only during walking. Here however, the robot recovers from pushes dynamically by finding proper foot place. Note that we give on-line state of the system to the navigator which corrects external disturbances.

For simple scenario of Fig.6.10 the robot is able to recover 40N which is far more than what we had in previous chapter 5, but less than static push recovery of chapter 4. Maximum recoverable pushes in this scenario with the same timings are 50N in left/right direction and 40N in back/front direction. In Fig.6.10 the left push causes self collision which we do not have any control over.

Remark that recovering pushes strongly depends on the state of the robot. If a push has the same direction with CoM speed, it becomes difficult to recover specially if it is too large. The role of constraints in navigator become important where one should take care of maximum step size and collision avoidance. Although we can constraint left foot for example not to go beyond right foot in right direction, this will reduce the ability of push recovery considerably. Because human character does this by passing swing foot above the other one to capture extra energy. This is in fact non-convex constraint which is out of the scope of this research. It also needs advanced arc trajectory design to avoid self collisions. Collision avoidance is in fact not addressed in any of 45, 53, 55 and 56.

However, we can deal with collision in a very limited way with current convex formulation of the problem. Assuming nominal minimum distance of feet $d_{\text{min}} = 0.13m$ and maximum foot step size of $step_{\text{max}} = 0.5m$, we add these two constraints to the optimization:

$$dir \times P_y[1] \geq dir \times P_y[0] + d_{\text{min}}$$

$$-step_{\text{max}} \leq P_y[1] - P_y[0] \leq step_{\text{max}}$$

Where $dir = \pm 1$ is a variable determining swing leg, used in Fig.6.3. With these constraints, foot can not pass over and steps can not be arbitrarily large to capture all extra energy in first step taken. Such limitation is in fact multiple step capturability strategy discussed in previous chapter. With these constraints, foot step planning becomes very limited of course and we are able to apply pushes at specific times only. For example strong left push could be applied only when swinging leg is the left one. The outcome of this scenario is shown in Fig.6.11 where left/right pushes are 30N and no collision occurs.

The scenario of Fig.6.11 is in fact limited, because constraints added to the
Figure 6.11: Push recovery scenario with $v_{ref} = 0$ and pushes of $30N$, lasting 0.5s each. Note that no collision occurs due to inclusion of preliminary collision avoidance constraints to the navigation problem.
problem are tough and the robot can recover form pushes only having specific timings.

After recovering pushes when there is no motion, we would like to test push recovery during walking as well. To this end, we first disable extra constraints, and then try pushes at maximum stable speed. This scenario is shown in Fig.6.12 with same pushes of 40N to different directions. Note that although there are two collisions happening, but feet overlap in very small regions. A suggestion to solve this problem is to add non-convex constraints. Side pushes are 40N at maximum while front and back pushes could be stronger depending on the speed of robot.

![Figure 6.12: Push recovery scenario with $v_{ref} = 0.4m/s$. Pushes of 40N each lasting 0.5s are applied to the system at $t = 30s$ to left, $t = 40s$ to front, $t = 50s$ to right, $t = 60s$ to back. Although there are 2 collisions happening (shown by black rectangles), but feet slightly collide which is solvable if we had non-convex optimization. Note that in this figure, $x - y$ scale is not true as robot’s rectangular feet are appearing to be square.]

In this section, we showed performance of the algorithm and its robustness against external pushes while doing in-place walking and also having maximum speed. Table 6.2 shows comparison of different push recovery scenarios:

<table>
<thead>
<tr>
<th>walking controller</th>
<th>chapter</th>
<th>left/right</th>
<th>front/back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stance</td>
<td>4</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>Captured</td>
<td>5</td>
<td>20</td>
<td>20+</td>
</tr>
<tr>
<td>Dynamic (0.0m/s)</td>
<td>6</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Dynamic (0.0m/s) + collision avoidance</td>
<td>6</td>
<td>30</td>
<td>30+</td>
</tr>
<tr>
<td>Dynamic (0.3m/s)</td>
<td>6</td>
<td>40</td>
<td>40+</td>
</tr>
</tbody>
</table>

The robot can recover pushes and maintain its path. We also discussed the problem of self collision and the fact that since we can not add non-convex constraints to our optimization, we are not able to deal with this problem with current
formulation. We tested the algorithm only in a limited scenario with specific timings where our optimization could solve additional restricting convex constraints. Dealing with self collisions systematically remains for future work.

6.3.3 Delayed communication

So far we had no argument on delays in communication. Before analyzing the effect of communication delays, we will first describe technically what happens between simulator and the controller process. Remember from chapter 2 that Atlas robot has a control loop of $1KHz$. So it publishes states (sensor position, velocity and torques) every $1ms$ and asks for commands to execute. If our ODE based simulator (Gazebo) was capable of simulating $1ms$ of real world simulation in less than $1ms$, then we could run real-time simulation. This depends on the power of computer. Note that our controller in simulation scenarios is in fact a separate process running on the same computer and exchanges messages with Gazebo using ROS packet based library.

On the current platform of simulation which is a quad-core Core i5 laptop running at $1.7GHz$ and using Intel HD 4000 graphics card, the simulation of Gazebo is $82\%$ real time without any controller involved. There is an option in Gazebo which enables it to wait for commands coming without executing simulation forward. If we do not use this option and our controller is capable of working at $200Hz$ for example, one commend is copied 5 times and executed in Gazebo for $5ms$. The controller process is triggered whenever a packet containing the state of robot arrives. During calculation of proper command for that packet, it neglects other incoming packets. Note that packets are also labeled by the time they are published. So by calculating time difference, we can find out the rate of our controller. But this does not tell us anything about delays between controller and Gazebo. By using the option in Gazebo, we can make sure that there is no delay, although the simulation gets very slow. Our controller in this chapter consists of 3 main tasks:

1. Low level controller quadratic optimization.

We have chosen SD/FAST and CVXGEN to achieve maximum possible speed in calculations. We also log everything inside our controller to the RAM memory of computer which has less delays. If we use Gazebo’s delay option, there is a parameter called $t_{expected-delay}$ which is the expected delay time for a command to come. When Gazebo publishes a state at time $t$, it simulates forward until...
If we choose $t_{\text{expected}} - \text{delay}$ to be very large, Gazebo certainly receives the command in between and always continues forward. However if we choose $t_{\text{expected}} - \text{delay}$ very small, Gazebo is always waiting for new command. This option is in fact used for synchronization of controller with Gazebo where some expected delay is assumed.

![Figure 6.13: Timing of different task with respect to the choice of variable $t_{\text{expected}} - \text{delay}$. The graph labeled "ROS-delay" is showing difference of time labels over state packets arriving at controller.](image)

We show timing of different tasks in Fig.6.13. Note that if Gazebo stops, other processes could run faster and thus lasting in smaller times. So different parameters are correlated when we simulate everything on the same machine. The controller almost runs in about $2.3 - 3.3 \text{ms}$ with all optimizations. $3.3 \text{ms}$ is when Gazebo is also running and consuming computational resources of the computer. With large choice of $t_{\text{expected}} - \text{delay}$, "ROS-delay" is about $7 \text{ms}$, indicating that it takes about $7 \text{ms}$ for a state packet to be published, received by controller and analyzed. If we assume the same delay $d$ from gazebo to the controller and vice-versa, $7 \text{ms}$ is $d$ plus one execution time which is around $3.3 \text{ms}$ regarding choice of $t_{\text{expected}} - \text{delay}$. Therefore, we can say $d \simeq 3.7$ and total time taken between when a state is published and its command is received is $2d + 3.3 \simeq 10.7 \text{ms}$. Almost all the scenarios we have discussed in this chapter and before are simulated with this delay. Note that according to Fig.6.13, if we choose $t_{\text{expected}} - \text{delay} \simeq 10.7 \text{ms}$, we get the most rapid
change in real-time factor. Below this value, the Gazebo is repeatedly stopped to receive proper command. Such small difference can easily make real-time factor dropping. Note also that maximum real-time factor when having controller inside the loop is 62% compared to the case of 82% when there is no controller. Maximum controller rate however when running on a similar separate machine is about $440Hz$ (corresponding to $2.3ms$).

The last thing worth to test in this part is the ideal case when there is minimal delay. We choose $t_{\text{expected-delay}} \simeq 3ms$ which is the case if the controller was running on board. With this choice, the robot can walk with $V_{\text{ref}} = 0.6m/s$ stably which was not the case before. However it can not go beyond this velocity even in ideal case which is therefore due to current parameter tuning.

### 6.3.4 System noises

In this section we want to add noise to different parts of the system and quantify its robustness. Noises are added to the following variables separately:

- Joint positions, velocities: Incoming sensor data may change system behavior. Note that our method uses a model which is updated by this data.
- IMU: Orientation, linear and angular velocities coming from IMU unit will actually affect orientation and foot position consistency.
- EOM: We add noise to each line of EoM and see how sensitive it is. Later we would also perform some model error analysis.
- Torques: We make output torques noisy to see how sensitive the method is to this variables.

Noises are added to the system by:

$$\hat{x} = x + N(0, \sigma) \quad (6.27)$$

Where $N(\mu, \sigma)$ denotes Gaussian noise function. We also test the system with $v_{\text{ref}} = 0$. Although one can perform similar analysis on different speeds, but the sensitive part is hybrid formulation and state machine which can make the system unstable. We have listed maximum tolerable noises in the system in Table 6.3. In this table, variables having - in front are not used in our algorithm and thus not sensitive to the noise.

Here we plot as an example, states of the system with existence of different noises on joint positions and IMU orientations. Compare the limit cycles of the system when the noise gets large; the robot does not fall, but stumbles considerably.
Table 6.3: Maximum tolerable noise. We perform a logarithmic search on $\sigma$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Noise strength $\sigma$</th>
<th>dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>position sensors</td>
<td>0.001</td>
<td>$rad$</td>
</tr>
<tr>
<td>velocity sensors</td>
<td>0.03</td>
<td>$rad/s$</td>
</tr>
<tr>
<td>torque sensors</td>
<td>-</td>
<td>$N.m$</td>
</tr>
<tr>
<td>IMU Orientation</td>
<td>0.001</td>
<td>$rad$</td>
</tr>
<tr>
<td>IMU Linear acceleration</td>
<td>0.001</td>
<td>$m/s^2$</td>
</tr>
<tr>
<td>IMU Angular velocity</td>
<td>0.003</td>
<td>$rad/s$</td>
</tr>
<tr>
<td>Translational lines in EoM</td>
<td>-</td>
<td>$N$</td>
</tr>
<tr>
<td>Rotational lines in EoM</td>
<td>1</td>
<td>$N.m$</td>
</tr>
<tr>
<td>Output torques</td>
<td>3</td>
<td>$N.m$</td>
</tr>
</tbody>
</table>

Figure 6.14: Comparison of limit cycles in the system with existence of different Gaussian noises in the system. "s" denotes $\sigma$ in the Gaussian of the form $N(0, \sigma)$. 

122
6.3.5 Model errors in link masses

In previous part, we investigated the effect of noise in various variables of the system. We know how random errors change limit cycles and disturb them. However, there are sometimes constant errors which cannot be predicted, measured or filtered. Modeling error is one of these problems, which is important in our model-based controller. It can happen in different parts of a model, like joint frictions, link masses, link sizes, softness of some link or soft contacts. Since we simulate the robot with a rigid body simulator, we are only able to simulate few of these aspects. In this part, we change some link masses in the simulator while they are kept intact in our SD/FAST model. These changes might make the robot heavier or asymmetric. Here we assume inertia matrix of a link to be the same as before.

![Figure 6.15: In 4 tests, we change the mass of links shown in a white box in above pictures: (a) pelvis, (b) l_uleg, (c) l_foot, (d) l_foot and r_foot.](image)

In Table 6.4 we have listed maximum extra mass that could be added to these links while in-place walking scenario is still stable.

Table 6.4: Maximum tolerable mass added to different links as model error.

<table>
<thead>
<tr>
<th>scenario</th>
<th>link name</th>
<th>maximum mass [Kg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>pelvis</td>
<td>3</td>
</tr>
<tr>
<td>(b)</td>
<td>l_uleg</td>
<td>5</td>
</tr>
<tr>
<td>(c)</td>
<td>l_foot</td>
<td>3</td>
</tr>
<tr>
<td>(d)</td>
<td>l_foot &amp; r_foot</td>
<td>3 each</td>
</tr>
</tbody>
</table>

Adding masses to the robot will in fact make the robot asymmetric while the model itself is still symmetric. We assume adding mass to all upper body links is equal to making the pelvis heavier. One might be interested to know what happens to limit cycles, which we show on Fig 6.16.

Following points are concluded from Fig 6.16.

123
Figure 6.16: In 4 tests, we change the mass of links shown in a white box in above pictures: (a) pelvis, (b) l_uleg, (c) l_foot, (d) l_foot and r_foot. We show nominal limit cycle by red color and actual limit cycle by blue. Shifts or skews are shown by black arrows and shrinks or scales are shown by green.

- (a) and (d): Symmetric extra load will affect average speed of the robot. Remember that we had no control over $x$ position and the robot had non-zero speed due to incremental errors. Since we do not have position feedback, we can observe the effect of systematic model errors better.

- (b), (c) and (d): Adding mass to the left of the robot reduces maximum $y$ speed (shrinking effect shown by green arrows). It means that the robot does not accelerate much to the left.

- (b) and (d): Although masses are added both to left leg in these plots, adding mass to upper parts skews the limit cycle differently compared to adding it to lower parts.

Our controller is in fact tolerating notable amount of mass added to links, but limit cycles have changed considerably. We can also perform similar tests during walking where $v_{ref} \neq 0$. Generally the maximum tolerable mass is less, but similar effects are observed.
6.3.6 Model error in link lengths

In this part, we are mostly interested in knowing how geometrical errors will affect the locomotion in our model-based method. We will have two scenarios:

- (a): make left leg longer than right one by enlarging the foot for 1.5 cm. This is equivalent to attaching a surface of 1.5 cm at the bottom of the shoe.
- (b): Moving knee joint 10 cm down in left leg.

Fig. 6.17 shows scenario (b) where the knee joint is shifted down. Note that for scenario (a) we cannot enlarge the foot more. Because before running walking controller, the robot does not stand and falls on right side due to its default joint position controllers.

![Diagram](image)

Figure 6.17: Diagram corresponding to test scenario (b) where the knee joint is shifted down for 10 cm in left leg. Ignore visualization discrepancies, since they are not important in optimizations.

Again we test the effect of these geometric errors with zero speed. The result is shown in Fig. 6.18.

The scenario (a) does not change limit cycles shape. In fact, the effect of this geometric change is that the robot detects touch down sooner while still it assumes there is no contact. This can disturb CoM positions slightly since the robot is compliant, but does not change hybrid state timings as they are calculated merely based on expected swing times (refer to Fig. 6.3). Thus we can see minor variations in the position of limit cycle for scenario (a).

In scenario (b) however, although the robot does not fall, we can see a skew in limit cycle’s shape. Variations are also quite smaller than scenario (a). This skew is similar to scenario (b) in previous part where we had changed thigh’s mass shown in Fig. 6.15. Shifting knee is somehow equivalent to making thigh link larger, which was done by changing mass before.
Overall, our algorithm is capable of tolerating large internal geometric errors unless task-space tracking is not affected considerably. The navigator is withstanding these errors by taking new steps which are assumed to be performed well by planner block.

6.3.7 Walking on rough terrain

So far we investigated various errors on the performance of our algorithm. These errors were noise, model errors, external pushes and delays. Among these tests, it was only push scenario to be intrinsically an external disturbance. In this part and the next one, we would like to investigate other types of external disturbances, appearing in the terrain. Here we test rough terrain which is random change while in next part we test slopes which are biased changes in the environment.

Our rough terrain is in fact hill-like, because if foot is placed on a step shape, it rotates considerably and disturbs the algorithm very much. We also denote terrain variations by $\delta$ which is the difference between maximum and minimum height. \(\delta\) is $10cm$ for this terrain compared to about $1m$ length of robot’s leg.

We give $v_{ref} = 0.2m/s$ to the robot and start on a hill. Fig.6.20 shows the performance of our controller which has the same configuration of parameters with Table 6.1. It speeds up on downhills while slowing down on uphills. It can also shift left or right if it finds a non-negative sideway component in gradient of the terrain. Note that one can achieve better performance by either choosing higher desired speed or using larger feedback gains. This will actually need tests to make sure that the locomotion is still stable.

Our test in this part is a demonstration of maximum performance achievable by current configuration of parameters. We do not aim at maximizing roughness or speed. In fact these metrics highly depend on tuning of parameters which is out
Figure 6.19: Picture of rough terrain test scenario with hills and valleys in front of the robot. Hills are shown with grass while valleys have soil color. Note that the robot has a position feedback which makes it sticking to straight direction ($y = 0$).

Figure 6.20: The performance of our controller with the same configuration of parameters as before. In this figure, the robot is given desired speed of $v_{ref} = 0.2m/s$. It speeds up on negative slopes and slows down when reaching an uphill. If gradient of the terrain has a component to sideways, it can cause the robot shift to the left or right and bypass a hill.
of scope of this research. In next part, we will ask the robot to go on gradually increasing slopes. We want to see what happens if slopes are not random like a rough terrain.

6.3.8 Walking on different slopes

In previous section, we had random slopes and roughnesses. Here, we want to give constantly increasing slopes and modify some parameters to achieve better performance. Note that principally, the environment/terrain is not perceived, meaning that we do not have any information from the environment and even we do not detect slopes by reading orientation of contacting foot. For the purpose of this scenario, the slope in front of the robot increases $0.02\text{rad} \simeq 1.14^\circ$ every $3m$. We gradually increase slopes since, placing foot on a discontinuity in the surface will change its posture, disturbing CoP calculations and stability of simulation in Gazebo. Fig[6.21] shows the simulation environment for this scenario.

![Figure 6.21: A series of slopes increasing gradually in front of the robot. The slope increases about 0.02\(\text{rad}\) or 1.14\(^\circ\) every 3m.](image)

Remember from previous part that the robot was speeding up or slowing down when reaching downhills or uphills respectively. One could think of using larger feedback gains, but still the terrain error is biased. To compensate such error, we add an integrator to the feedback formulation of Eqn[6.24] An integrator on speed will mean position feedback:

\[
v_{\text{ref}} = \left[ v(t) + (v(t) - \dot{P}_{\text{CoM}.x}) \times 0.5 + (x(t) - P_{\text{CoM}.x}) \times 0.5 \right] - \dot{P}_{\text{CoM}.y} \times 1.0 - P_{\text{CoM}.y} \times 0.5
\]  

(6.28)

Where \(v(t) = 0.2m/s\) and \(x(t) = 0.2t\) for the purpose of this test. Again, one can use larger feedbacks here, but this prevents the robot from responding flexibly to environment, such as shift appearing in Fig[6.20] for instance. The output of this scenario is shown in Fig[6.22] where the robot reaches slope of 4.6\(^\circ\) at maximum.
In this scenario, it is remarkable that with a very simple position feedback, the robot can go up much more compared to normal configuration of parameters and feedback rules we had in previous scenarios (where it goes up to 2.3° maximally). Maximizing performance is again a matter of parameter tuning, out of scope of this research.

### 6.4 Summary

In this chapter, we proposed a controller for the robot that made walking locomotion completely dynamic, i.e. with assumption of having no feet. We moved optimization from continuous-time states (in planner) to discrete-time states (in navigator). The planner block became a series of PD controllers, following desired trajectories compared to previous chapter where we had MPC control. The low level block is also kept as before (see Fig.4.1). After simplification of the model and deriving linear relations between states of the system at hybrid phase changes, we proposed a foot-step planning method that minimizes future steps and state of the robot, given a desired speed and an ideal relative position of foot steps. The outcome of this optimization is therefore next foot location for swing leg to touch down. It is then the planner that has to generate proper arc and swing to that location.

We tested our algorithm for different desired speeds and various errors and unwanted effects in the environment. External pushes were rejected online by adjusting foot locations. In the same way, the robot was able to withstand internal noises in the system, model errors in mass or length of links and also external disturbances such as slopes in the terrain. We have also tested all scenarios, assuming fixed 10ms of delay which is relatively high compared to a practical case where there is no simulation and the controller is running on-board.

With the new method, although it is not possible to perform exact foot placement like previous methods, but the robot recovers from various errors and unwanted
phenomena. This method rejects disturbances dynamically and does not strongly depend on big-size feet like many other humanoid robots. It is also applied to a relatively heavy robot, with the same dimensions as human and also relatively strong actuators. Within optimizations, we take care of various physical constraints such as actuator torque limits and friction cones which make the approach promising to run on a real robot. Calculations are also quite fast on simple computer and take few milliseconds per time step. Also with various simplifications of the model, we leave very few number of parameters to tune, compared to other methods. In next section, we will wrap up the report and discuss various aspects remained open for future work.
Chapter 7

Conclusion and future works

In this chapter, we compare the finalized method introduced in chapter 6 with previous versions (in chapters 4 and 5) and summarize the procedure followed to reach this controller. We will also enumerate advantages and disadvantages of this controller. At last, a list of important works remaining for future is proposed.

In chapter 2, we reviewed various methods and algorithms for controlling legged robots. The first group of these approaches was model-free, i.e. without any knowledge about the robot. These methods are usually optimized off-line to have best/desired performance when running. Although one can include perturbations to the optimization and make the parameter set robust with the cost of reducing performance slightly, but the method usually becomes locally robust or optimal and may not work for different conditions. The advantage of these methods is therefore low computational resources needed online.

The second group of methods relied on kinematic model of the robot where one could determine a Cartesian position given joint angles and vice versa. This information have been widely used in many robots, either floating based or fixed based. But the algorithm is not compliant and can not have precise fast motions easily, since there is an intrinsic trade off between tracking and compliance which is usually determined by feedback gains. These methods are however more precise than previous group and of course being moderate in terms of computations.

The third group however rely on a dynamics model of the robot which takes mass and inertia properties of the robot into account. Using this data, one can predict future motion of the robot, reduce feedback gains and produce actuator inputs that comply with the dynamics of the robot. Therefore better tracking is usually expected from this methods while having better interaction with the environment. Compliance is important, because the robot should not be harmful for itself and environment as well. This group of methods is however computationally heavy and is therefore recently developed and improved due to advancements in computational units.
In this work, we have chosen a method from the third group since we were targeting fast biped locomotion which needs dynamics knowledge of the robot to have compliance and tracking. We were also aiming at considering various constraints in the control loop for which, whole body optimization method was suitable.

In chapter 3, we described the platform of simulation we are working with. The robot under control is a human-sized robot, being torque controlled. It is simulated in an ODE based simulator and connected to the controller using a packeting protocol called ROS. We use SD/FAST dynamical model of the robot which is implemented in C and known to be very fast in terms of computations. We also use CVXGEN optimization library which is used for quadratic convex optimizations and generates specific codes that minimize amount of calculations. Principally, most of elements in the matrices appearing in a standard quadratic problem are zero and CVXGEN avoids multiplying them. All these powerful packages and libraries are used to develop the method.

We formulated the whole body optimization for our robot in chapter 4. Given desired Cartesian accelerations for center of mass and feet, this controller generates proper feed-forward torques to be sent to the robot. We also added some more blocks controlling inputs and also made posture control tracking upright posture all the time inside low level controller. The architecture of control blocks is shown in Fig. 7.1 in which the block we formulated in chapter 4 is called low level controller.

![Figure 7.1: Structure of integrating different parts. The low level controller is responsible for generating joint torques, given Cartesian accelerations. The planner generates these accelerations given desired foot positions. On top of all, the navigator plans steps in future.](image)

Afterwards, in the same chapter we proposed very simpler Cartesian controllers to investigate various aspects of the low level controller. The last planner was generating static walking locomotion which relied on having support polygons in both double support and single support phases. However this method was very fragile and producing artificial motion, having large sagittal (left/right) variations for center of mass.

In chapter 5, we tried to solve this problem by predicting the motion of robot in single support phase. To this end, we simplified the robot into a linear inverted
pendulum model and also introduced the concept of capture points used to make the simplified model stable. Using this idea, we assumed there is no foot in single support phase providing ankle torque and thus the robot is like a pendulum falling. In double support however, we used support polygons and formulated a model predictive controller to track a desired capture point which lead to stability after few steps. The planner block was therefore composed of this MPC controller together with foot arc trajectory controllers. In navigator block, we derived closed form equations which determined desired capture point, given that the robot stops in $N$ steps. The problem of sagittal motion was rather solved by future predictions. The control method was not completely statically-stable anymore, as CoM could sometimes fall out of support polygon. However the robot was not able to reach speeds more than $0.2m/s$, because of being statically-stable in double support.

In chapter 6, we released the assumption of having foot in any phase and thus made the control completely dynamically-stable. We also removed double support phase as we assumed there is no ankle torque for control. The robot was principally always falling, but navigator could plan next foot location for swing foot to touch down and this method captured extra energy in the robot. Thus, the locomotion problem was reduced to proper foot placement problem using simplifications we made on the robot model and walking patterns. The planner therefore only generated accelerations corresponding to a linear inverted pendulum falling, and also arc trajectory following. It was the navigator that optimized foot steps and gave next desired foot location, using a discrete-time model predictive controller. Compared to previous method, the new one optimized future foot steps while this sequence was obtained in close form before, having no online flexibility to perturbations. We can summarize all methods in Table 7.1.

Table 7.1: Comparison of walking methods proposed in chapters 4, 5 and 6. Bold items show QP optimizations.

<table>
<thead>
<tr>
<th>Method</th>
<th>chapter</th>
<th>static speed</th>
<th>capture points</th>
<th>foot steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum speed</td>
<td>4</td>
<td>0.06m/s</td>
<td>0.2m/s</td>
<td>0.5 m/s</td>
</tr>
<tr>
<td>low level controller</td>
<td>whole body</td>
<td>whole body</td>
<td>whole body</td>
<td></td>
</tr>
<tr>
<td>planner (CoM)</td>
<td>PD</td>
<td>MPC/LIPM</td>
<td>LIPM</td>
<td></td>
</tr>
<tr>
<td>planner (foot arcs)</td>
<td>PD</td>
<td>-</td>
<td>fixed sequence</td>
<td></td>
</tr>
<tr>
<td>navigator</td>
<td>-</td>
<td>fixed sequence</td>
<td>MPC</td>
<td></td>
</tr>
</tbody>
</table>

A simple PI controller determined desired speed to be given to the navigator block of our method in last chapter. We had thus simplified the robot so that to be controllable even with such simple controller like a wheeled robot.
7.1 Advantages, drawbacks

The final controller is able to withstand large perturbations either externally from pushes or terrain variations, or internally from noises, model errors or delays. It can also reach higher speeds compared to its earlier versions. We can enumerate advantages of our proposed final controller as:

- **Compliance**: The robot does not react to external pushes strongly and adapts itself, proven by push recovery test in chapter 4. It can also change weight to left and right almost without any double support in between. Contact forces have minimal spikes in this case which shows compliance.

- **Model abstraction**: Total of 35 degrees of freedom are guided by 2 task-space movements.

- **Cartesian control**: We can have control over foot and CoM in Cartesian space which improves navigation and obstacle avoidance.

- **Push recovery**: We reject strong pushes dynamically by adjusting steps. This could not be done by our preliminary methods based on support polygons. (Compare push recovery in chapter 5 and 6).

- **Tolerating delays**: Commands could have delays up to 10ms which is far worse than in practice on a real robot.

- **Fast calculation**: All calculations take 2 – 3ms per time-step compared to where they re-plan every 100 time-steps or after each contact change.

- **Withstanding noises**: Random noises can be rejected easily. Although we do not have any filtering on sensor readings, our optimizations can rejects them by planning. Note that filtering introduces delay and makes control problem difficult.

- **Model errors**: Although the method is based on dynamics model, we showed that asymmetric masses or link lengths do not cause the robot fail easily.

- **Fast locomotion**: The final controller can reach high speeds which could be even improved, if assuming less delayed communication.

- **Rough and sloped terrains**: The robot can walk on considerably rough terrain or slopes without having perception.

During different tests, we were mostly looking for showing the behavior of our algorithm. The only drawback in our finalized method is exact foot placement.
Although the first step could be imposed in navigator, but the method needs more freedom for future steps to recover and capture extra energy. One can also combine this method with its former versions for such goal which is out of scope of this research.

## 7.2 Future works

So far, we have achieved most of our goals: compliance, fast walking, push recovery and robust un-perceived locomotion. However our method can be improved regarding following aspects:

1. **Steering**: With the current formulation, from the odometry discussed in chapter 3 to final PI controllers used on top of the navigator, we assume that the robot is always walking forward. One should also add steering capabilities to make locomotion more flexible.

2. **Improved LIPM**: While simplifying the robot, we assumed point contact and no inertia in LIPM which is not the case for the simulated robot, since its body has inertia and hip torques correct for posture error. One can also think of making model abstractions more precise, including hip torques.

3. **Contact modeling**: For higher speeds, full contact assumptions of feet could be violated. One can model them more precisely and thus have better control on contacts.

4. **Orientation control**: We assume fixed back and arm joints while they can be used to make motion more natural and help for posture control and stabilization.

5. **Better tuning**: Although we have very few parameters to tune, specially for arc trajectories of feet, we have tuned them manually. One can propose better tunings to achieve more precise tracking.

6. **Transferring to real robot**: Our method performs calculations in the order of 2 – 3ms which could be done on-board. So we seek for transferring this method to a real humanoid too.

These improvements are needing time or reformulation of mathematical equations which are beyond the limited time of this research.
Bibliography


