Robust Balancing for Bipedal Robot via Model Predictive Control

by Michael Napoli
Advisor: Dr. Ayonga Hereid

The Ohio State University
Department of Mechanical and Aerospace Engineering
July 2022

Thesis submitted in partial fulfillment of requirements for a degree of Bachelor of Science in Mechanical Engineering with Research Distinction from The Ohio State University

Acknowledgments

I would like to thank the Ohio State University Department of Mechanical and Aerospace Engineering for their support and guidance, as well as for awarding me the research grant which enabled this undergraduate thesis.

Abstract

Robust balancing controllers are essential for bipedal robots to safely operate in real-world applications where human-robot interactions are a common practice. While the balancing controllers being developed are effective, they struggle when adjusting to untested motions and environments. Popular controllers commonly rely on heuristic techniques, and simplified models of the intended system, and are optimized to compute applicable joint inputs quickly. What they sacrifice in robustness, they often make up for in computational efficiency and speed. Here, the triple pendulum model is used as a unique method of simulating the dynamics of a bipedal robot in the 2-D saggital plane. The goal of this research is to develop a control architecture which can stabilize the triple pendulum in real time using the linear center of mass dynamics, and when introduced to random initial conditions, fluctuating stance heights and external disturbances. These objectives will be achieved via a model predictive control architecture, supplemented by the angular linear inverted pendulum model and an inverse dynamics function which computes the applicable low-level joint torques. Various optimization algorithms, most notably the nonlinear Newton's optimization and the nonlinear gradient descent algorithm, will also be tested with the intent of running in real-time. The initial algorithm design stage was completed in MATLAB and Python, before being implemented in the MuJoCo simulation system in Python for final testing. Most notably, the simplified model could be simulated for a prediction horizon of length 20 with a time-step of $0.05[s]$ ($1[s]$ of look ahead time) with an average calculation time of $363.13[ms]$. As was expected, the largest drawback to implementing the discussed control system is the computation time required for each call of the optimization program. That said, results show that implementing the MPC system would result in more stable overall performance, and creates a system which can enter new environments with little-to-no tuning while maintaining stability.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ALIP	Angular linear inverted pendulum
MPC	Model predictive control
NNO	Nonlinear Newton's optimization
NGD	Nonlinear gradient descent
ID-QP	Inverse dynamics-quadratic programming
CP	Capture point
PID	Proportional integral derivative
RL	Reinforcement learning
NN	Neural network
HZD	Hybrid zero dynamics
LQR	Linear quadratic regulator
FMPC	Fast model predictive control
QP	Quadratic programming
PH	Prediction horizon
OC	Optimal control
SC	Sub-optimal control
LIP	Linear inverted pendulum
COM	Center of mass
ZMP	Zero moment point
CMP	Centroidal moment point
SOC	Stochastic optimal control
MEM	Modified euler method
FOOC	First-order optimality condition
NRM	Newton-Raphson method
SISO	Single input, single output
MISO	Multiple input, single output
BLS	Backtracking line search
ODE	Ordinary differential equation
RIST	Random initial state test
HVT	Height variation test
EDT	External disturbance test

List of Symbols

Symbol	**Meaning**
x	Objective function inputs
$f(x)$	Objective function
$\nabla f(x)$	Gradient of the objective function
$\nabla^2 f(x)$	Hessian of the objective function
ε	Zero approximation tolerance
$A(x)$	Equality constraints
$B(x)$	Inequality constraints
α	NGD step-size coefficient
$g(t,\gamma)$	Arbitrary dynamics function
γ	Dynamics function state variables
h	MEM time-step
q	TPM state variable
q_{ref}	TPM reference trajectory
u	ID-QP optimized joint torques
s	ALIP state variable
s_{ref}	ALIP reference trajectory
v	MPC-optimized inputs to the ALIP state
P	Length of the prediction horizon
t'	Prediction horizon time-step
$C(s,v)$	MPC general cost function
$C_{lcb}(s,v)$	Logarithmic cost barrier function
$C_{cp}(s,v)$	Capture point cost function
w	Assorted gains for the MPC cost function
x_c	x-Position of TPM COM
L	Angular momentum about the ALIP ankle joint
L_c	Angular momentum about the ALIP COM
u_a	Torque about the ALIP ankle joint
z_c	Constant height of the COM
m_c	COM magnitude

Symbol	**Meaning**
θ	TPM link positions
τ	TPM low-level joint torques
δ	Capture point angular momentum limit
d_{max}	Stability region range about the foot
X	ID-QP function variables
z	ID-QP function inputs
J	ID-QP state Jacobian
σ	ID-QP cost function gains
h_c	Desired height of the COM
θ_c	Desired orientation of the TPM torso link

Chapter 1

Introduction

To people disturbances are commonplace, but where human intuition prevails in maintaining balance, bipedal robots are not always so advanced. For them to enter environments where human-to-robot interaction will be normalized, they must have robust performance when met with sudden deviations from equilibrium.

Real-world scenarios and disturbances are challenging for the stabilization of bipedal robots. This is due to their dimensional complexity and inherent instability, and while the balancing controllers being developed are effective, they struggle when adjusting to untested motions and environments. Modern controllers commonly rely on heuristic techniques, and simplified models of the intended system, and are optimized to compute applicable joint inputs quickly. What they sacrifice in robustness, they often make up for in computational efficiency and speed.

The triple pendulum model (TPM) is seen as a unique method of simulating 2-D bipedal dynamics in the saggital plane. Here, it is used as a stepping stone to larger 3-D bipedal systems such as the Digit robot designed by Agility Robotics. It should be noted that the link orientation referenced as a desired state in later sections somewhat models the stance of the Digit robot.

The goal of this research is to create a low-level controller which can stabilize the TPM using a real time model predictive control (MPC) architecture. This will be achieved with the assistance of the angular linear inverted pendulum (ALIP) model and a quadratic programming-based inverse dynamics (ID-QP) function which converts the optimized ALIP inputs into the applicable TPM joint torques. Various optimization algorithms will also be tested; most notably the nonlinear Newton's optimization (NNO), and nonlinear gradient descent (NGD) algorithm.

The controller will also be simulated in three primary testing scenarios; the random initial state test, the height variation test, and the external disturbance test. Through these examinations it will be proven that the strength of the intended architecture is in the robustness of its recovery range, and its fast application to new systems when supplied with an accurate and comprehensive state space.

It is also important to note the use of a unique cost function which incorporates a simplified form of the capture point (CP) method. This research is conducted on the premise that the MPC structure can more robustly stabilize a bipedal robot without complicated and tedious tuning methods or training time. A lack of training time means faster application to new environments, and more robust results when entering fields where data may be difficult, or impossible to obtain accurately. Possible applications of the controller include, but are not limited to, rehabilitation services, facility oversight and maintenance, courier servies, etc.

Chapter 2

Literature Review

2.1 Existing Balance Controllers

As robotic applications increase in popularity, the safety and robustness of the humanoid system has become a sort of limiting reagent to them being integrated into the workforce. While their application is apparent, the need for them to maintain robust balancing has become the focus of numerous companies, universities, and independent research studies. At the beginning of the thesis process, several control structures were evaluated for their robustness and application with the intent of selecting a novel approach to the proposed control problem. The methods considered consist of proportional integral derivative (PID) control [1, 2, 3], reinforcement learning (RL) [4, 5, 6], and most recently model predictive control. Since MPC is the architecture chosen for this thesis, it will be discussed in more detail in Section 2.2.

A common architecture used in industry is the proportional integral derivative controller. PID controllers create an efficient closed loop system and, when tuned properly, can follow desired joint configuration paths with a high degree of accuracy. That said, they have no understanding of future states, and require a large amount of tuning when being introduced to new environments. There is research in the field of optimizing the gains of PID controllers, most notably in [2], but the systems discussed must be re-optimized when difficulties are observed, which can be a tedious and lengthy process. For this reason, PID controllers are not ideal for a bipedal robot whose intent is to be capable of entering any variation of environment at a given point in time.

More recently, the study of artificial intelligence and machine learning have lead to the development of the reinforcement learning approach to controlled motion. These methods are considered model-free, which means they do not explicitly reference the underlying physics of bipedal walking. Instead, the RL architecture is used to award an existing neural network (NN) based on the actions taken in the previous state. This introduces the need for model training and, depending on simulation accuracy, may not be transferable to real systems post-simulation. In [4], Castillo introduces hybrid zero dynamics (HZD) directly into the neural network. In this approach the NN calculates the desired trajectory based on the HZD and uses an adaptive PD controller to compute the necessary torques in each of the robot joints that best follow the trajectory given.

2.2 Model Predictive Control

MPC has quickly become the dominant control architecture for legged robots in recent years [7, 8, 9, 10, 11, 12]. Poised as one of the only structures which can account for future configuration states, MPC has shifted into use primarily as a trajectory planning algorithm with PID and RL controllers calculating joint torques in order to follow the desired path. In this research, a simplified linear model based on the angular momentum about the center of mass will be used as the desired trajectory inputs (see Sect. 4.2.1), and then shifted to joint torques using a comprehensive inverse dynamics optimization problem (Sect. 4.4).

A powerful architecture commonly used in stabilizing industrial processing systems, the initial formulation of MPC was as the linear quadratic regulator (LQR) problem and was created by Rudolf Kalman in the early 1960s [11]. This concept was popularized because of its potential for an infinite time horizon, but had little impact on control technology due to its inability to account for inequality constraints. In the 1970s, with the addition of the necessary constraint protocols, the LQR formulation was re-titled MPC. In these applications, the inequality constraints were accounted for and so the industrial use of the MPC problem grew rapidly.

Recently, MPC has been used as the foundation for many optimal controllers in legged robotics because of its simulation of future states and costs [8, 9]. With a forward thinking system, the biped can make better decisions on what it should do in the current state, so as to not jeopardize its stability down the line.

The largest inhibitor to this control method is the time required to calculate the optimal set of inputs. For this reason, multiple techniques are taken from Wang's paper on *Fast Model Predictive Control (FMPC)* [10]. Here, the MPC problem is formulated as a quadratic programming (QP) problem, and a culmination of common approaches are used to create a system that can run in real-time over the full model dynamics. These techniques consist of warm-starting, move blocking, extreme iteration limits, and exploitation of the QP problem structure.

Warm-starting is the action of using the previous solution of the optimization problem as an initial guess to the current MPC formulation. While very simple, this is objectively the most impactful method implemented, as it decreases the number of required steps appreciably. FMPC also describes the use of move blocking, which works to hold the input constant over a set number of prediction horizon (PH) windows, allowing the solver to reach farther into the future without increasing the number of optimized variables. This method is effective in larger degree of freedom systems, but is not as dramatic as the previous approach.

Finally, the MPC solver is given extreme iteration limits. In most optimization algorithms the maximum iteration break is a sign that the solver could not find an optimal solution. In this instance, and because of the use of warm-starting, it can be reliably assumed that the solver is initially very close to the solution. With this in mind, the maximum iteration is set to a very low value (between 5 and 10). While this increases the likelihood of premature breaks, and thus sub-optimal solutions, it is understood that the desired variables being pushed will be updated in the next time step.

The primary method used to catalyze the problem solution in FMPC is the exploitation of the QP formulation. With the intent of building the system from scratch, and utilizing more complex cost barriers than the QP structure allows, this approach was not implemented. Instead, the MPC problem is formulated as a nonlinear optimization problem, which is solved via the nonlinear newton's optimization algorithm, and the nonlinear gradient descent algorithm discussed in Section 2.3, and formulated in Chapter 3. Although this implementation leads to inherently slower calculation time, it is shown that the controller is still capable of performing to the desired standards. The simplified model, which was implemented to further increase computational efficiency, will also be discussed in Section 2.4, and formulated in Section 4.2.1.

2.3 Unconstrained vs. Constrained Optimization

In controller theory there are two overarching types of policies; optimal and sub-optimal control (OC, and SC) [13]. In the case of sub-optimal control, the solution found is often accurate, fast, and achieves a given purpose, but does so in an inaccurate, or sloppy manor. The most common implementation of this can be seen in PID controllers. PIDs can be tuned to a high degree of accuracy, but often have a tendency to fail when given a disturbance they were not designed for. OC policies on the other hand are implemented using a weighted cost function which allows them to make informed decisions based on the current and future states. If the function is time-dependent, it is considered an open-loop controller, and is otherwise considered a closed-loop controller. Here, the ladder of the two is utilized.

The cost functions constructed are often composites of quadratic and logarithmic functions, allowing for guaranteed smooth curves. This is important since the search algorithms discussed in Sections 3.2 and 3.3 are dependent on the assumption that the curves are smooth at all points. The search for the optimal solution to an OC is commonly coined optimization, and comes in two forms; unconstrained and constrained. In the context of MPC, optimization is discussed in the papers [12, 14, 15] among many others.

Unconstrained optimization is the formulation of the OC with no boundary or motion constraints. This is relatively straight forward compared to its constrained counter-part, and is commonly used to vet the solution to systems before implementing the applicable boundaries. Constrained optimization on the other hand implements equality and inequality constraints which have higher risk of creating local minimum/maximum points, and/or saddle points. Since optimization works via locating the point where the derivative of a function is approximately zero, the existence of these points can cause the solver to produce sub-optimal solutions. This topic will be more thoroughly discussed in Chapter 3.

2.4 Angular-Linear vs. Linear Momentum Models

Since its derivation in 2001 [16], the linear inverted pendulum (LIP) model has been the most popular method of simplifying the center of mass (COM) dynamics for a walking bipedal robot. By treating the COM of the robot as a point mass, and connecting it to the ground via a variable length pole, the motion of the COM can be approximated accurately. If the deviation of the COM from the center point is small, it is reasonable to make the assumption that the height is constant. This key assumption also makes the dynamics linear, allowing for fast simulation speed.

These equations are derived by using the linear momentum about the COM, and have been instrumental in the development of many present day controllers. With the dynamics being linear, they can be simulated exponentially faster. Perhaps more importantly though, the LIP model makes the zero moment point (ZMP) and the centroidal moment point (CMP) very easy to calculate. Allowing the controller to calculate an optimal step length for walking.

In [17], the LIP model is further expanded to utilize the angular momentum about the pivot point as a state variable, and the angular momentum about the COM as an input variable. This allows for a better representation of the upper-body dynamics of the robot, as well as supports the idea that angular momentum about the pivot is a better method of quantifying stability than linear velocity (as in the LIP model). The model derived in [17] will be referred to as the angular linear inverted pendulum (ALIP) model.

In the controller designed here, the ALIP model will be used as a stepping stone to calculating the TPM inputs. The MPC system will first select an optimal set of angular momentum values for the length of the PH. Then, an inverse dynamics system, to be discussed in Section 4.4, will be used to convert the desired momentum to low-level joint torques.

Chapter 3

Background Derivations

Before discussing the formulation and analysis of the MPC problem, some background in optimization is presented. Stochastic optimal control (SOC) is a more zoomed in view of OC, where the solutions being generated are assumed to be approximations of the optimal solution. There are multiple different methods to achieve this. Here, the nonlinear Newton's optimization algorithm, and the nonlinear gradient descent algorithms will be derived for use in the MPC system discussed in Chapter 4. The algorithms and methods discussed in this chapter are not unique to the research conducted, but are derived for clarity and practice.

With the intent of use in the MPC architecture, the concept of minimizing optimization will be the focus of the following derivations, but the optimization algorithms discussed are applicable to maximization systems as well. The modified euler method (MEM) will also be evaluated as a method for solving initial value problems for linear state spaces with generally faster computation time.

3.1 Constrained Optimization

In constrained optimization, the goal is to find some set of inputs which minimize a given objective function and while satisfying a predetermined set of constraints.

The notation and formulation shown here is derived from [15], which discusses the goal of the Newton's root finding algorithm with respect to optimization. First, the general optimization problem is stated formally.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & A(x) = a,\ B(x) \leq b \\ & x \in \mathbb{R}^N \end{aligned} \tag{3.1}$$

Where $f(x)$ is the objective function, x is a list of real number inputs of size N, and $A(x)$ and $B(x)$ are the equality and inequality constraint functions respectively. For the sake

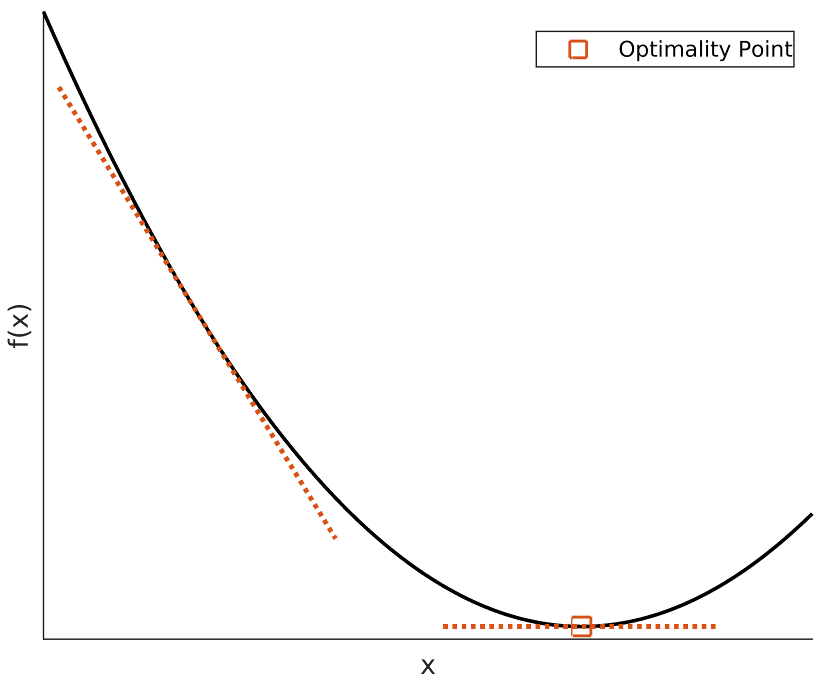

Figure 3.1: Derivative Trend for General Quadratic Function

of convenience, the general form of the optimization problem will be used to formulate the algorithms discussed in Sections 3.2, and 3.3.

Next, the solution of the minimization problem must be identified. Figure 3.1 shows an arbitrary quadratic function with the derivative marked at two points by tangent lines. By comparison, it is clear the global minimum is found at the point where the derivative of $f(x)$ is equal to zero.

$$\frac{d}{dx}(f(x)) = 0 \tag{3.2}$$

In multivariate systems, the derivative becomes the gradient, or the partial derivative of the objective function with respect to each of the individual inputs. Because of the problematic nature of machine precision, it is common to take the norm of the gradient and compare to some small threshold approximation of zero. In this case, equation 3.2 becomes:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^N \tag{3.3}$$

$$||\nabla f(x)||_2 \leq \varepsilon \tag{3.4}$$

Where N is the number of inputs being evaluated, and ε is the applicable approximation of zero. The norm used here is the *L2-norm*, but other norm methods are also suitable. Equation 3.4 is hereby referred to as the first-order optimality condition (FOOC). The following sections will discuss two of the most common methods used to locate this value.

3.2 Nonlinear Newton's Optimization

Nonlinear Newton's optimization is the foundation of many optimal controllers. The general-form solution of the NNO algorithm with constraints is presented, and notes on its performance can be found in Section 5.1. Some comparisons between NNO and the opposing nonlinear gradient descent algorithm will also be discussed.

The Newton-Raphson method (NRM) was first created as a root finding algorithm with the intent of taking meaningful steps towards the root (zero point) using the Newton-step equation, stated in equation 3.5.

$$x_{i+1} = x_i - \frac{f'(x_i)}{f(x_i)} \tag{3.5}$$

Where x_i is the current guess, $f(x_i)$ is the objective function at the current guess, and $f'(x_i)$ is the derivative of the objective function. These steps would be repeated until some threshold for zero, ε, was passed. See the pseudocode below for a more demonstrative layout.

Algorithm 1 Newton-Raphson Method

```
Require: $x_{initial}$, $f(x)$, $f'(x)$
Ensure: $x_{root}$
 1: function NEWTONRAPHSON($x_{initial}$)
 2:     $x \leftarrow x_{initial}$                    ▷ Initialize Loop Variables
 3:     $y \leftarrow f(x_{initial})$
 4:     while $y \geq \varepsilon$ do
 5:         $\dot{y} \leftarrow f'(x)$
 6:         $x \leftarrow x - \frac{\dot{y}}{y}$                ▷ Newton-step
 7:         $y \leftarrow f(x)$
 8:     end while
 9:     return $x_{root} \leftarrow x$
10: end function
```

General NRM can be used to find the roots of single input, single output (SISO) equations which are continuous and cross the x-axis. Later, the algorithm was expanded to utilize the gradient to solve for an equation with multiple inputs and a single output (MISO). These algorithms are commonly used as an alternative to solving systems analytically and are very powerful.

With the inclusion of the second derivative, the NR algorithm can be similarly used to search for where $\frac{d}{dx}f(x) = 0$. In the case of a MISO system, the second derivative is the partial derivative of the gradient with respect to each input, and is commonly called the Hessian.

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N x_1} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix} \in \mathbb{R}^{(n \times n)} \tag{3.6}$$

Where $\nabla^2 f(x)$ denotes the second partial derivative of the objective function with respect to each input. It is important to note that the Hessian is symmetric across the diagonal, which allows for the calculation to be somewhat simplified during implementation. The Hessian creates an extremely accurate step-size to the iterative structure of the NNO algorithm, but is dimensionally unfavorable. As the number of inputs increase, the size of the matrix grows exponentially.

Once the Hessian is formulated, the step towards the FOOC point is as follows.

$$x_{i+1} = x_i - \nabla^2 f(x_i)^{-1} \times \nabla f(x_i) \tag{3.7}$$

Where $\nabla^2 f(x_i)^{-1}$ is the inverse of the Hessian matrix at the current guess. Note that since the Hessian is positive semi-definite for any objective built from quadratic functions, it is always invertible. Equation 3.7 will be hereby referred to as the Hessian-step.

Because the algorithm is meant to be used in real-time, more thorough break statements will be reported in the pseudocode for algorithm 2.

Algorithm 2 Nonlinear Newton's Optimization

```
Require: $x_{initial}$, $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$
Ensure: $x_{minimum}$
 1: function NEWTONSOPTIMIZATION($x_{initial}$)
 2:     $x_0 \leftarrow x_{initial}$;
 3:     $y_0 \leftarrow f(x_{initial})$;
 4:     $iter \leftarrow 0$;
 5:     while $y_i > \varepsilon$ do                          ▷ Break Code: 0
 6:         $g \leftarrow \nabla f(x_i)$;                     ▷ Gradient Calculation
 7:         if $||g||_2 < \varepsilon$ then                ▷ Break Code: 1
 8:             break;
 9:         end if
10:         $H \leftarrow \nabla^2 f(x_i)$;                   ▷ Hessian Calculation
11:         $x_{iter+1} \leftarrow x_{iter} - H^{-1} \times g$;   ▷ Hessian-step
12:         $y_{iter+1} \leftarrow f(x_{iter+1})$;
13:         $iter \leftarrow iter + 1$;
14:         if $iter \geq max\text{-}iter$ then            ▷ Break Code: -1
15:             break;
16:         end if
17:     end while
18:     return $x_{minimum} \leftarrow x_{final}$
19: end function
```

The implemented algorithm will have break statements for maximum iteration count and zero cost, in addition to a break for the FOOC. Each break is accompanied by the appropriate code, which tells the user what condition was used. The zero cost break is only possible in cases where the objective function is guaranteed to be positive or zero. This is suitable for the MPC system described here because the cost function is built from a sum of quadratic error functions, and thus never falls below zero. This also allows the algorithm to avoid calculating the gradient and Hessian when not required, further increasing its efficiency.

In most systems, the maximum iteration break is used to identify infeasible regions. This allows the optimizer to break the search and retry with a new initial guess, as opposed to wasting time looking for a solution where one does not exist. Because of the use of warm-starting (discussed in Section 2.3), the iteration limit is set to a very low number, as was described in [10].

3.3 Nonlinear Gradient Descent

Nonlinear gradient descent is a common method for large system optimization. It is primarily seen in the machine learning discipline, and is characterized by small steps, and slow convergence towards the optimal solution.

The NGD algorithm structure runs parallel to that of NNO, with the exception of the need for the Hessian matrix. For the purposes of the NNO algorithm, the Hessian can be described as the most precise step-size necessary for each of the inputs to find their optimal value. In NGD, the Hessian is replaced by a scalar step-size coefficient multiplied by the gradient, a directional vector that points towards the optimal solution.

$$x_{i+1} = x_i - \alpha \cdot \nabla f(x_i) \tag{3.8}$$

Where α is termed the step-size coefficient. Equation 3.8 will be hereby referred to as the alpha-step. In many systems a variable step-size coefficient is used as opposed to a constant scalar, the most popular algorithm being backtracking line search (BLS) [18]. Algorithm 3, describes this process fully.

Note that this is a suboptimal step-size coefficient. It is only large enough such that it decreases the current cost, implying that the step-size is very small, and innaccurate. That said, each iteration of the BLS loop only requires a single call to the objective function, as opposed to the Hessian which currently requires N^2 function calls.

The intent behind using NGD as opposed to the NNO algorithm was solely based on the lack of need of the Hessian matrix. Without this, each iteration of the search loop would be exponentially faster and, ideally, the number of optimized variables could be increased without heavily impacting runtime. The pseudocode for the NGD algorithm is shown in algorithm 4.

Algorithm 3 Backtracking Line Search

```
Require: $x_{current}, \alpha_{initial}, \gamma, f(x), \nabla f(x)$
Ensure: $x_{BLS}, \alpha_{BLS}$
 1: function BACKTRACKINGLINESEARCH($x_{current}, \alpha_{initial}, \gamma$)
 2:     $\alpha_0 \leftarrow \alpha_{initial}$;
 3:     $y_0 \leftarrow f(x_{current})$;
 4:     $g \leftarrow \nabla f(x_{current})$;
 5:     $iter \leftarrow 0$;
 6:     while $y_{iter} \leq \varepsilon$ do
 7:         $\alpha_{i+1} \leftarrow \gamma \cdot \alpha i$;
 8:         $x_{new} \leftarrow x_{current} - \alpha_{i+1} \cdot g$;
 9:         $y_{i+1} \leftarrow f(x_{new})$;
10:         $iter \leftarrow iter + 1$;
11:     end while
12: end function
```

As can be seen, the only major difference between algorithm 2 and algorithm 4 is the calculation of the Hessian matrix. The two algorithms are compared in Section 5.1 for speed and robustness.

Algorithm 4 Nonlinear Gradient Descent

```
Require: $x_{initial}, \alpha, f(x), \nabla f(x)$
Ensure: $x_{minimum}$
 1: function GRADIENTDESCENT($x_{initial}, \alpha$)
 2:     $x_0 \leftarrow x_{initial}$;
 3:     $y_0 \leftarrow f(x_{initial})$;
 4:     $iter \leftarrow 0$;
 5:     while $y_i > \varepsilon$ do                          ▷ Break Code: 0
 6:         $g \leftarrow \nabla f(x_i)$;                      ▷ Gradient Calculation
 7:         if $||g||_2 < \varepsilon$ then                   ▷ Break Code: 1
 8:             break;
 9:         end if
10:         $x_{iter+1} \leftarrow x_{iter} - \alpha \cdot g$;   ▷ Alpha-step
11:         $y_{iter+1} \leftarrow f(x_{iter+1})$;
12:         $iter \leftarrow iter + 1$;
13:         if $iter \geq max\text{-}iter$ then               ▷ Break Code: -1
14:             break;
15:         end if
16:     end while
17:     return $x_{minimum} \leftarrow x_{next}$
18: end function
```

3.4 Modified Euler Method

A fundamental process to the MPC architecture is the simulation of a given dynamical system over a desired prediction horizon. At the beginning of the implementation process, the MATLAB *ode45()* function was used for its robustness and accuracy. That said, through testing of the MPC system, it was made clear that the *ode45()* function was slower, and more complex than what was needed for the linear system dynamics discussed in Section 4.2.1.

For this reason, the modified Euler method of solving ordinary differential equations (ODE) was implemented to increase the simulation speed. It will also be shown that, with an appropriate time step, the ODE solver is suitable for larger, nonlinear systems as well.

To start, the explicit Euler method makes the assumption that the system dynamics are linear for small changes in time.

$$\gamma_{i+1}^{Eu} = \gamma_i + h \cdot g(t_i, \gamma_i) \tag{3.9}$$

Where γ_i is the current state of an arbitrary dynamics function, $g(t_i, \gamma_i)$, and h is the time step. This results in an inaccurate approximation of the state γ_{i+1}^{Eu}, which is supplemented by the MEM equation below.

$$\gamma_{i+1} = \gamma_i + \frac{h}{2} \cdot \left(g(t_i, \gamma_i) + g(t_{i+1}, \gamma_{i+1}^{Eu})\right) \tag{3.10}$$

The results of equation 3.9 are used to approximate the derivative of the final position. This is then compounded with the derivative of the current state to account for nonlinear changes in the dynamics along the path. The result is a more accurate approximation of the derivative at the current state, and thus a more accurate simulation. The pseudocode for a single step in the simulation process can be found below.

Algorithm 5 Modified Euler Method

```
Require: $t_0$, $\gamma_0$, $h$, $g(t, \gamma)$
Ensure: $\gamma_f$
1: function ModifiedEuler($t_0$, $\gamma_0$, $h$)
2:     $t_f \leftarrow t_0 + h$
3:     $\dot{\gamma}_0 \leftarrow g(t_0, \gamma_0)$
4:     $\gamma_{Eu} \leftarrow \gamma_0 + h \cdot \dot{\gamma}_0$
5:     $\dot{\gamma}_f \leftarrow g(t_f, \gamma_{Eu})$
6:     $\gamma_f \leftarrow \gamma_0 + \frac{h}{2} \cdot (\dot{\gamma}_0 + \dot{\gamma}_f)$
7:     return $\gamma_f$
8: end function
```

Where $\dot{\gamma}$ is the results of the dynamics function at a given state and time step, i.e. the derivative of the current state. This process is then repeated for the windows of the simulation frame.

Chapter 4

Problem Formulation

In this chapter, the fundamental components and equations used in the main controller will be derived. The equations and formulas discussed here are considered the primary tools used to achieve the project's overarching research goals.

The culmination of the methods discussed in this chapter create a feedback control loop which works via the process flow diagram shown in figure 4.1. This diagram will be used as a road-map of sorts for the remaining sections. These sections are as follows; model predictive control, modeling techniques (consisting of the ALIP model, the TPM, and a brief note on the Digit system), the capture point method, and the inverse dynamics formulation.

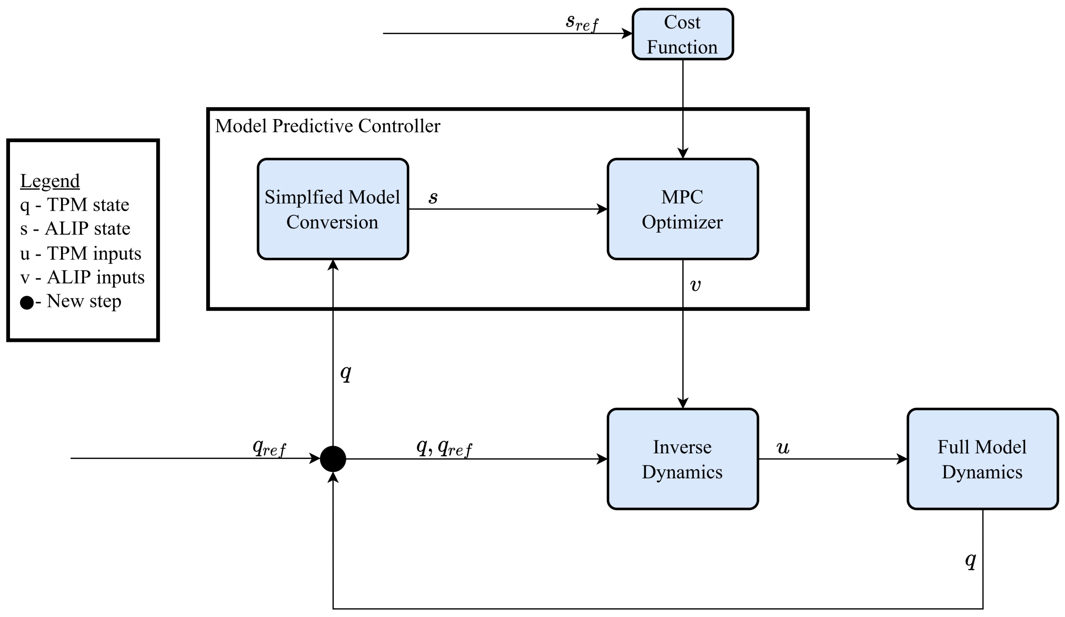

Figure 4.1: MPC Feedback Block Diagram

The initial state, represented by the black dot, consists of the current TPM state, q, and reference trajectory, q_{ref}. These are converted to the simplified ALIP model variable, s, before being funneled into the MPC optimizer which calculates the desired input, v, using the reference trajectory, s_{ref}, and a predefined cost function. This is converted back to the low-level joint torques, u, via the inverse dynamics function discussed in Section 4.4 and implemented. The process is then repeated until a force quit is issued by the user, or the system breaks.

The variables q, u, s, and v will be more thoroughly defined in Section 4.2.

4.1 Model Predictive Control

As previously stated MPC is an optimal control architecture which has become very popular in the field of legged robotics. It uses a dynamic model of the system to predict future states, select the most effective set of inputs, and repeat the process in the next time-step. This decision-making process is represented well in figure 4.2.

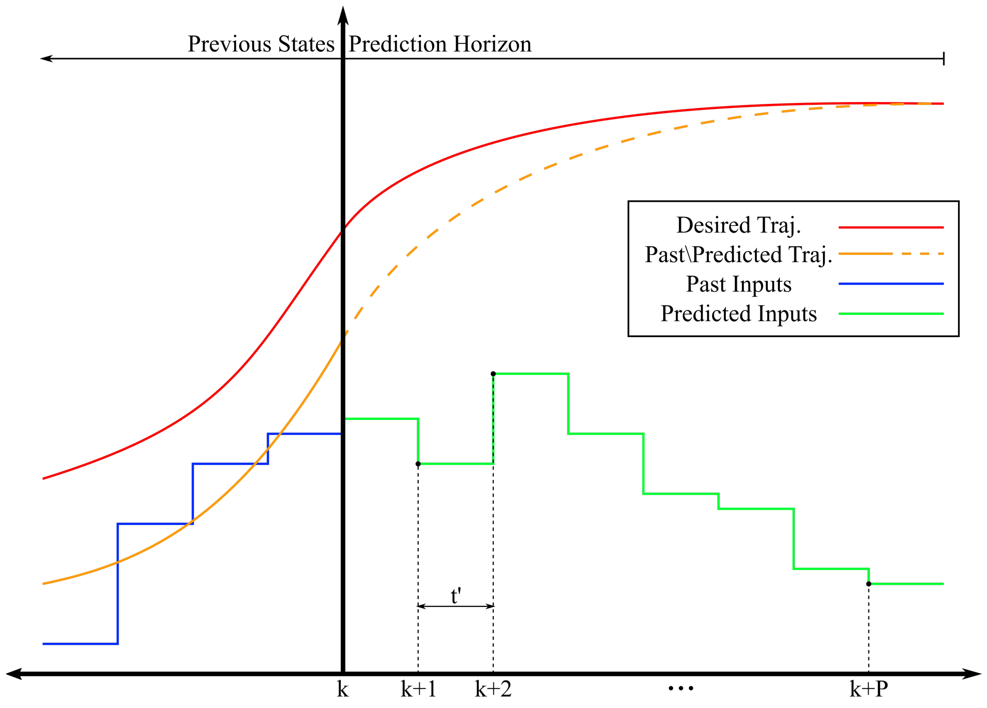

Figure 4.2: MPC Decision Structure

The MPC formulation starts at the black line notated by k on the horizontal axis and precedes to simulate states for each window of the prediction horizon, separated by the time-step, t', until it reaches the desired count, P. At each knot of the horizon, it evaluates the cost of the state according to the reference trajectory, as well as the cost associated with the input at that knot. It then adjusts its inputs if necessary until the cost function is at a local minimum. The first input in the series is then implemented into the controller, and the process is repeated in the next time-step.

The most accurate mathematical representation used to solve the MPC problem is a minimizing optimization problem. In this case, the future states and inputs become subject to a tunable cost function which is then forced to obey equality and inequality constraints similar to the systems derived in Section 3.1. With that in mind, equation 3.1 will be redefined in terms of the MPC structure.

$$\begin{aligned} \underset{s,v}{\text{minimize}} \quad & C(s,v) \\ \text{subject to} \quad & A(s,v) = a,\ B(s,v) \leq b \\ & s,v \in \mathbb{R}^n \end{aligned} \tag{4.1}$$

Where $C(s,v)$ is referred to as the cost function, and is a function of s, a list of states simulated over the length of the prediction horizon using v, a list of inputs for each window of the PH. The cost function and inequality constraints are further broken down into their applicable components in equation 4.2.

$$C(s,v) = C_{lcb}(s,v) + C_{cp}(s,v) + \sum_{k=0}^{P} \left[w_s (s_{d,k} - s_{a,k})^2 + w_{v,1} (\Delta v_k)^2 \right] \tag{4.2}$$

Where C_{lcb} is the barrier function defined in equation 4.3, s_d is the desired state, and s_a is the actual state. The only non-constraint cost applied to input is magnitude change to prevent the inputs from making dramatic and unnecessary jumps. w_s and $w_{v,1}$ are the state and input cost gains respectively. Finally, C_{cp} is the cost of the capture point formulation, which will be more thoroughly discussed in Section 4.3.

In place of the standard inequality constraint formulation, a logarithmic cost barrier (LCB) is implemented so that the inputs do not exceed a certain maximum value. The use of logarithms is also helpful in that it does not negatively impact the smooth curve as do other absolute value-based methods.

$$C_{lcb}(s,v) = w_{v,2} \sum_{k=0}^{P} \left[\log(v_{max}^2) - \log(v_{max}^2 - v_k^2) \right] \tag{4.3}$$

Where v_{max} is the maximum allowable input to the system, and $w_{v,2}$ is the maximum input gain. For both equation 4.2 and 4.3, the cost is summed over each window of the PH. The values of each gain can be found in table 4.1.

Gains	Values
w_s	2500, 10
$w_{v,1}$	100, 0
$w_{v,2}$	1
w_{cp}	100

Table 4.1: Table of Cost Gain Values

It is important to note that w_s and $w_{v,1}$ have two gain values. This is because the state space being used here is comprised of two state and input variables and will be discussed in Section 4.2.1. w_{cp} is also included here for clarity, but is derived in Section 4.3.

4.2 Modeling Techniques

4.2.1 Angular Linear Inverted Pendulum Model

After a considerable amount of testing in full model optimization, it was deemed necessary to simplify the COM dynamics of the TPM to a linear system. Currently, the primary method of achieving this is to utilize the linear inverted pendulum model which was originally derived in [16]. This method exploits the simplified COM dynamics for walking pattern generation in a 3-D bipedal robot. The method is initially used to specify walking speed and direction, but has since become the backbone of many optimal controllers. By linearizing the COM dynamics, the system is capable of being simulated extremely efficiently, allowing for the expansion of the PH in the MPC problem formulation.

According to [17], the LIP model can be further adjusted to include a state variable for the angular momentum about the pivot point. This creates a more robust controller as it also introduces the angular momentum about the COM as a designated input to the system. The components to the ALIP system can be seen below.

$$s = \begin{bmatrix} x_c \\ L \end{bmatrix} \quad v = \begin{bmatrix} L_c \\ u_a \end{bmatrix} \tag{4.4}$$

Where s is composed of the state variables for the ALIP model, and v consists of the inputs. More specifically, x_c is the horizontal location of the COM, L corresponds to the angular momentum about the pivot point, L_c is the angular momentum about the COM, and u_a is the torque about the pivot. These state equations are then used to rearrange the standard LIP model state space to create:

$$\dot{s} = \begin{bmatrix} \frac{1}{m_c z_c} & \frac{\dot{z}_c}{z_c} \\ 0 & m_c g \end{bmatrix} s + \begin{bmatrix} \frac{-1}{m_c z_c} \\ 1 \end{bmatrix} v \tag{4.5}$$

Compared to the LIP model, equation 4.5 maintains a better understanding of the momentum required for balancing, and can be similarly decoupled in the frontal and saggital planes for 3-D modeling. By use of this simplification, the MPC problem is able to be completed much more quickly, making the potential for real-time implementation more realistic. The controller is also capable of increasing the length of the prediction horizon, taking additional future kinematic states into account.

That said, because the MPC problem is now calculating inputs that do not directly correspond to joint torques, the controller requires an additional step before publishing desired torques to the robot's driving motors. This conversion is completed via an inverse dynamics optimization problem described more thoroughly in Section 4.4.

4.2.2 Triple Pendulum Model

It is important to note the use of the triple pendulum model, a system of three links with varying lengths and masses. For the purpose of implementation the TPM was utilized for optimization algorithm development. The algorithms and methods intended for the full size implementation would be first developed for the TPM.

An example of the TPM system and its desired orientation can be seen in figure 4.3. This configuration was chosen because of its similarities to the Digit ankle, knee and hip joints. It was also important to test if the system could maintain equilibrium at non-zero resting positions. In these cases the inputs calculated to each link are also non-zero.

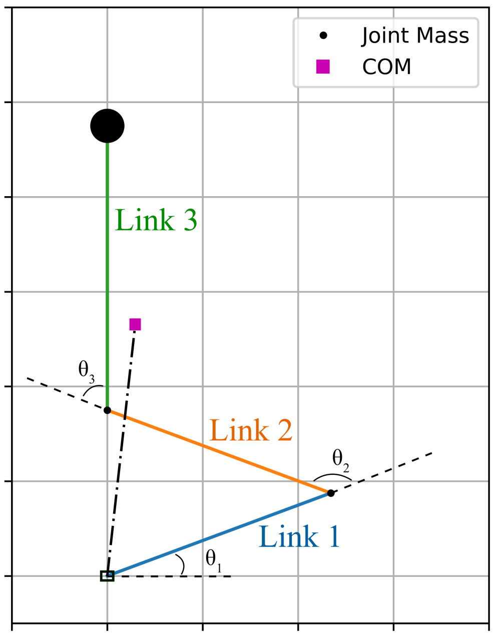

Figure 4.3: TPM Example - Near Desired Configuration

This model was meant to serve as a two-dimensional representation of a biped in the saggital plane. It is composed of three actuated joints, has a fixed position at the ankle joint, and has an unequal distribution between the link's lengths and masses. Here, Link 1 represents the connection between the ankle and the knee joint, Link 2 is the knee to hip joint, and Link 3 is the torso.

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \quad \dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{4.6}$$

Each link is composed of two state variables, the angular position, θ, and velocity, $\dot{\theta}$. These are compiled into a single state variable, q, which takes the joint torque inputs, u, shown in equation 4.7.

$$q = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad u = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{4.7}$$

The input variable, u, is composed of a torque for each of the actuated joints in the TPM. In the context of the control diagram referenced previously, these are outputted from the ID-QP function and are optimized to track the desired L_c trajectory.

4.2.3 Digit Model

While the bipedal robot, Digit, was not simulated with the control system discussed due to a lack of project time, it is still noteworthy to reference its application.

The intent of this research was to design a control architecture which could be easily expanded to a bipedal system. The TPM was used as a stepping stone for algorithm design as it was similar in structure to the 2-D orientation of Digit, seen in figure 4.4.

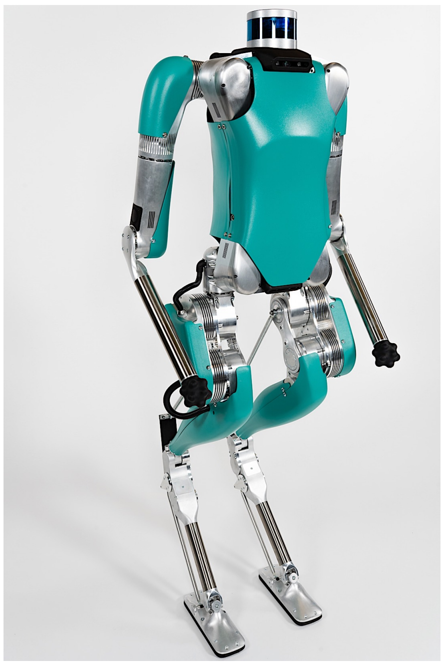

Figure 4.4: Stock Photo of Digit Robot

In the first period of testing, a comprehensive state space for the Digit was used to optimize its 20 actuated joints over a set PH. With the large number of control parameters, and due to the fact that the dynamics of the joints are nonlinear, it was deemed too computationally expensive to run in real time. At which point it was decided to go the route of the ALIP model and map the desired angular momentum to the joint torques as a post-MPC step. Ideally, in order to be converted to a Digit model system, the only two components from the block diagram in figure 4.1 that would need to be adjusted are the simplified model approximation function, and the inverse dynamics function. These adjustments will be discussed more in Chapter 6.

4.3 Capture Point

The capture point, derived thoroughly in [19, 20], is a method of evaluating the stability of a humanoid robot based on the orientation of the LIP model and the implemented joint torques. In the capture point method there are three possible outcomes when a biped is introduced to a disturbance. It can 1). use solely its joint torques to maintain balance, 2). take a step towards the ZMP to catch itself, or 3). fall. These three option's ranges are calculated using a support polygon which encompasses a geometric area around the biped's feet.

While stepping is an essential part of stability, this project will only address the first and third situations listed above. The controller will make the assumption that stepping is not a possibility. This is similar to if the robot were standing on a step-ladder, near a wall, or at the top of a set of stairs and in this case will be used to simplify the project.

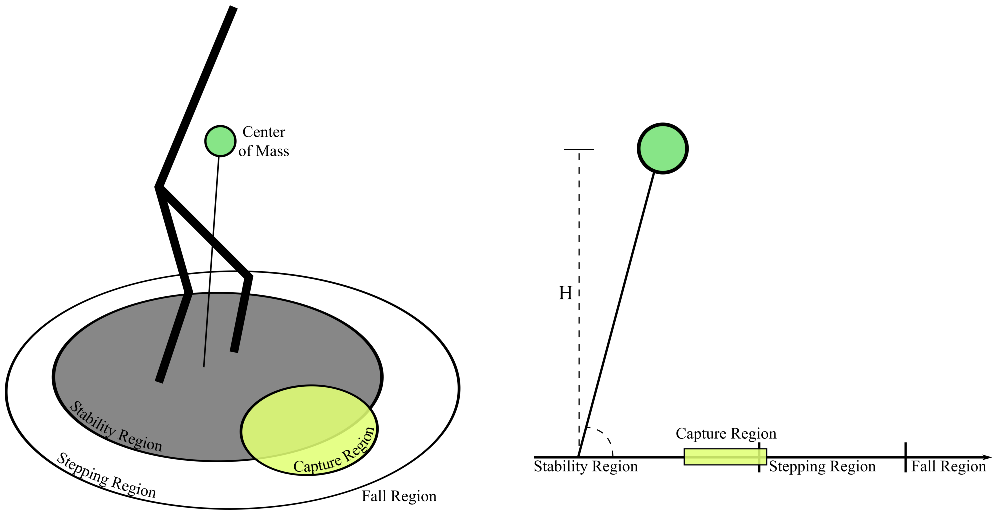

Figure 4.5: Capture Point Regions Demonstration

In the figure, the three regions of the capture point are represented by circles around the base of the biped. The yellow capture region is the calculated region which the robot must account for to stay upright. It is understood that if the capture region remains in the gray stability region, it requires only the necessary joint torques to maintain balance. With this in mind the cost function implements a barrier surrounding the edges of the support polygon's stability region.

It should be noted that the paper referenced in [8] utilizes a more fundamental implementation of the capture point method into the cost function for a MPC, similar to that which is being done here. The results exhibited are considerable for balancing on one foot and in various other highly unstable configurations.

The equation for the physical location of the capture point is subject to the dynamics being used for modeling. The ALIP model is used in conjuncture with the capture point dynamics to create the following.

$$\zeta = m_c g d_{max} \tag{4.8}$$

Where ζ represents the capture point limit, m_c is the magnitude of the COM, g is the gravity force, and d_{max} is the length of the foot in either direction. This limit represents the maximum allowable angular momentum about the pivot, and is then incorporated into a logarithmic cost barrier as follows.

$$C_{cp}(s,v) = w_{cp} \sum_{k=0}^{P} \left[\log(\zeta^2) - \log(\zeta^2 - L_k^2)\right] \tag{4.9}$$

Where L_k represents the angular momentum about the pivot, and is supplied by the ALIP state space variable, s. In this formulation, the variable ζ creates a simplistic representation of the stability region for the TPM. This is also accompanied by the gain w_{cp} marking its priority as mentioned in Section 4.1.

4.4 Inverse Dynamics

At the foundation of the ALIP model is the use of the angular momentum about the COM as a control variable. This allows for a more accurate appreciation of the underlying upper body dynamics of a bipedal robot. That said, it is difficult to analytically convert the MPC optimized angular momentum directly to TPM inputs. To account for this, a secondary optimization problem is performed which converts a desired angular momentum, L_c, to a set of low-level joint torques, u.

This optimization problem exploits the quadratic programming structure; a separate optimization structure from the algorithms discussed in Chapter 3. The notation and structure used here is restated directly from [21].

$$\begin{aligned} \underset{X \in X_{ext}}{\text{minimize}} \quad & ||\dot{J}_z(q,\dot{q})\ddot{q} + J_z(q,\dot{q})\dot{q} - \ddot{z}||^2 + \sigma W(X) \\ \text{subject to} \quad & D(q)\ddot{q} + H(q,\dot{q}) = Bu + J^T(q)\lambda \\ & J(q)\ddot{q} + \dot{J}(q)\dot{q} = 0 \end{aligned} \tag{4.10}$$

Where X is a list of cost function variables, and in this case are the acceleration of the applicable joints, and the joint torque variables. The variable z represents the desired configuration of the simplified model, and J is the Jacobian at a given model position. $W(X)$ is the cost function, σ is a list of applicable cost gains, and D, H, B, and λ are all components of the equality constraints. It should be noted that for all components of the objective function, only $W(X)$ is non-constant, and thus is the only portion being minimized. Here, X and z are structured in the format below.

$$X = \begin{bmatrix} \ddot{q} \\ u \end{bmatrix} \quad z = \begin{bmatrix} h_c \\ \theta_c \\ L_c \end{bmatrix} \tag{4.11}$$

Where h_c is the height of the COM, θ_c is the torso orientation measured from the horizontal x-axis, and L_c is the angular momentum about the COM optimized by the MPC. It should also be noted that through the MPC problem, the x-position of the COM, x_c, is also optimized. The angular momentum calculated by the MPC solver contains gains which move the COM toward the zero position. This causes the inverse dynamics function to indirectly track x_c with a relatively high degree of accuracy considering it is not explicitly stated.

In practice, the QP problem can be completed extremely quickly, and does not negatively impact the performance of the controller to a large degree, if at all. Because of the unique combination of inverse dynamics and the QP structure, this implementation will be referred to as ID-QP, similar to the [21] notation.

Chapter 5

Results and Discussion

The following chapter evaluates the results of the controller. Initially, the NNO and NGD algorithms were compared using an idealistic ALIP simulation and the most efficient was chosen for full-system implementation. Once the necessary algorithm was selected, testing for the controller was split into three parts; a random initial state test, a height variation test, and an external disturbance test.

5.1 Algorithm Comparison

This section details the comparisons between the NNO and NGD algorithms as implemented in the Python language. The algorithms were tested with the MPC architecture discussed, but the conversation will revolve around their computation time with systems of increasing dimensionality. It should also be noted that the model used for testing was an ideal ALIP model, meaning the implementation error between the controller and the actual system were close to negligible. This allowed for consistent results in the MPC and thus better comparison between the two algorithm's performances.

For each algorithm, the parameters of the MPC controller were held constant for a simulation of 50 randomly generated initial positions offset from equilibrium. Once the sample was complete, the prediction horizon was incremented by one, and the process was repeated until the length of the prediction horizon was 20. Each window of the prediction horizon was $0.05[s]$ long, making the maximum window reach $1.0[s]$ into the future. From these data points, the average necessary number of iterations, the average minimum iteration time, and the average calculation time for an optimal set of inputs were compared and are expressed in figure 5.1.

It is shown that the calculation speed for an individual iteration is exponentially smaller for the NGD algorithm, but is impaired by the large number of steps needed to find the optimal solution. For this reason, the NNO outperforms the NGD algorithm for nearly every length of prediction horizon because of the extremely accurate step size coefficient

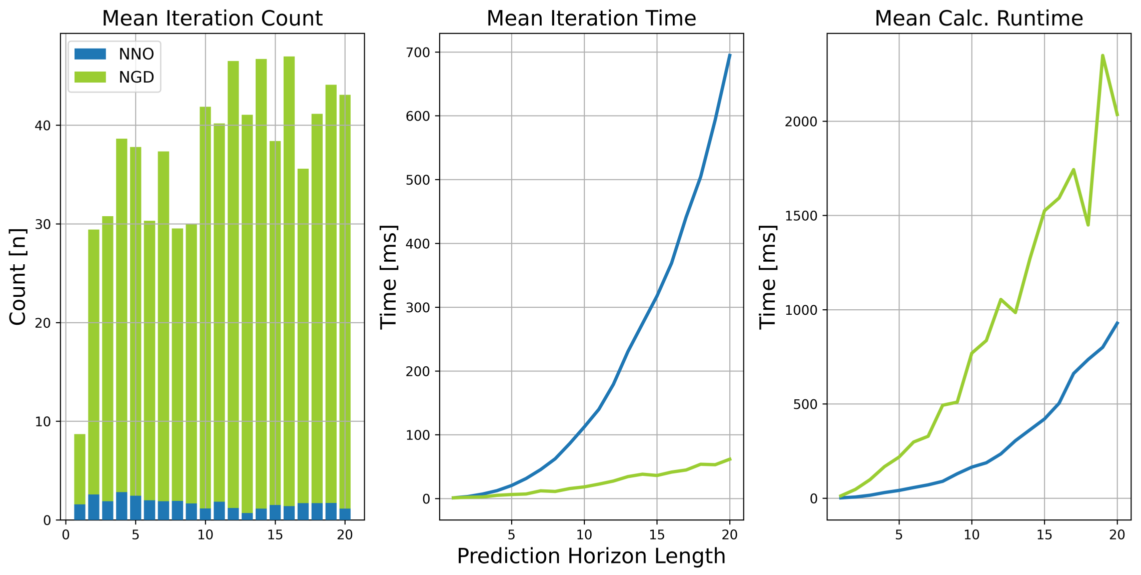

Figure 5.1: NNO vs. NGD Runtime Efficiency

which the Hessian creates.

That said, it was shown through testing that the NGD algorithm behaved more consistently for cases where the state space approached a cost barrier. In these situations, the cost would very easily approach infinite, causing for the iteration steps to become less predictable. Figure 5.2 shows the break codes for the NNO and NGD algorithms respectively.

There are five possible break codes each algorithm is able to meet the conditions for. These consist of the divergence break (-2), iteration break (-1), zero cost break (0), FOOC met (1), and zero change break (2). The optimal conditions are 0 and 1, because they imply the system is either at the stability point, or is approaching it accurately. The zero

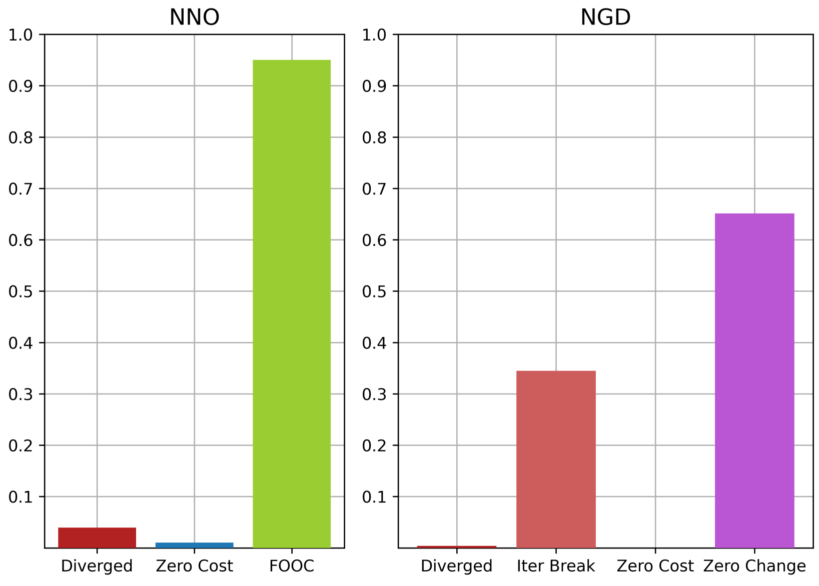

Figure 5.2: Simulation Break Frequency Comparison

change break implies the algorithm's step size fell below a certain threshold and it was unrealistic to continue iterating. An iteration break (-1) means the number of optimization steps exceeded the maximum allowed, and, finally, the divergence break (-2) means the system was not able to find a solution. The divergence break condition is met when the objective function returns either *nan* or inf. Ordinarily this means the system was not able to recover from a disturbance and approached a cost barrier where the output was inf.

As can be seen from the figure, NNO had both the best and worst performance in this area. It found the optimal solution 95.01% of the time it was executed, but also diverged 3.95%. That said, it could be argued that the conditions generated here were more dramatic than those that a real system would encounter. Furthermore, while NGD executed iteration breaks in 34.47% of the simulations, it greatly outperformed NNO in divergence trends with only 0.38% of the runs breaking.

In general, the system performed extremely well considering that, out of the total number of simulations, only 4.33% broke due to the inability to find an optimal solution. This may sound large, but it should be noted that 1). the system is restricted from stepping, meaning it is impossible to recover when in the stepping region of the support polygon, 2). the initial positions generated here were relatively large deviations compared to what the actual system experiences, and 3). finding an optimal solution for small lengths of the PH is inherently more difficult as the controller can not see very far ahead of its current state.

Based on these results, the NGD algorithm was used during the initial testing stages, when the system was being configured and was more likely to have faults. NNO was implemented once the performance was ideal and with the intent of improving the overall calculation time.

5.2 Triple Pendulum Model

Once the algorithms were tested and evaluated to be working, the MPC was implemented into the TPM using the ID-QP function. There were three primary tests, each evaluating the stability of the controller and its run-time efficiency. These included randomly selected initial states, adjusting the desired height mid-simulation, and applying an external disturbance after the system reached equilibrium. These tests are discussed in the following sections.

For all testing setups the configuration of the MPC system and the TPM were held constant. For the MPC architecture, the PH was set to 20 windows with a time-step of $0.05[s]$ giving a total look-ahead time of $1.0[s]$. The TPM had links that measured $0.5[m]$, $0.5[m]$ and $0.6[m]$ for links 1, 2 and 3 respectively. The mass for each joint was set to $5[kg]$, $5[kg]$ and $30[kg]$ for link 1, 2 and 3 respectively. In the ID-QP function the desired torso orientation was set to a constant $\frac{\pi}{2}[rad]$ and the height, with the exception of the height variability test, was set to $0.9[m]$.

5.2.1 Random Initial State Test (RIST)

RIST was used during initial testing of the inverse dynamics function. Once the program was started, an initial state was generated using the Python *numpy* library. The range of the angle deviations were $-0.1 \leq \Delta\theta \leq 0.1[rad]$ from the equilibrium position. As well as a randomly generated angular velocity between $-0.1 \leq \dot{\theta} \leq 0.1[rad/s]$. This procedure was primarily to test the general performance of the controller during the initial stages of design, and the cost gains for the various functions were tuned until performance was acceptable.

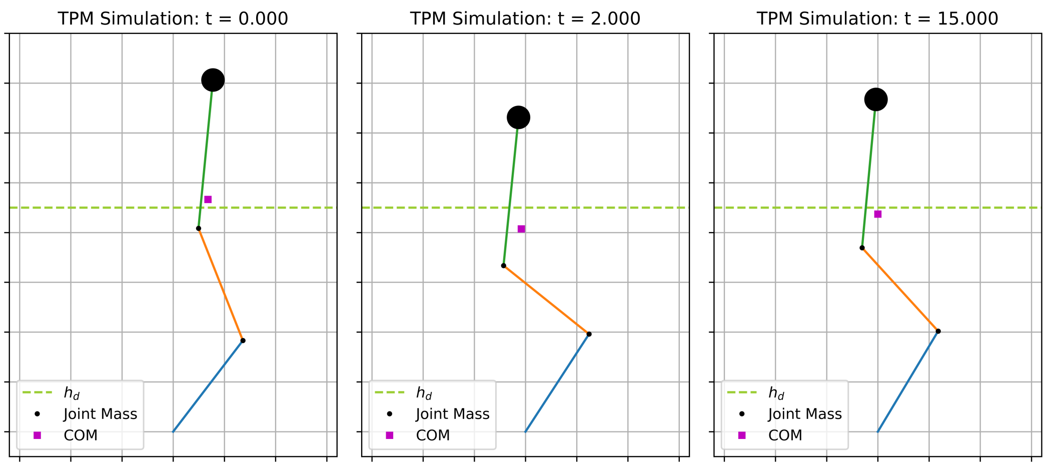

Figure 5.3: HVT - Three Primary TPM Positions

This environment propagated relatively dramatic initial positions which would often approach the bounds of the ALIP model constraints. That said, the system would rarely not be able to reach stability in this setup. The results here are an isolated test of the TPM recovery system.

Figure 5.3 shows the TPM at three points of the simulation. The first is the initial, randomly generated position. This test was isolated relatively arbitrarily, but shows an example of a state where the COM was lying close to the stability region bounds as stated in Section 4.3. The second state is the pendulum mid-recovery, and the third and final image is the TPM in its resting position.

Figure 5.4 demonstrates the low-level joint torques calculated from the ID-QP function which achieve the angular momentum selected from the MPC controller. It is shown that the computed torques oscillate heavily, having a potentially negative impact on what would be real motors. This oscillation was somewhat decreased by implementing gain costs on the magnitude of the torques calculated from the ID-QP function.

The maximum torque allowed by the joints was set to $250[Nm]$ and was maintained by the controller. Future tests can work to limit the oscillations seen by modifying the appropriate ID-QP gains and will be discussed in Chapter 6. While the TPM position and

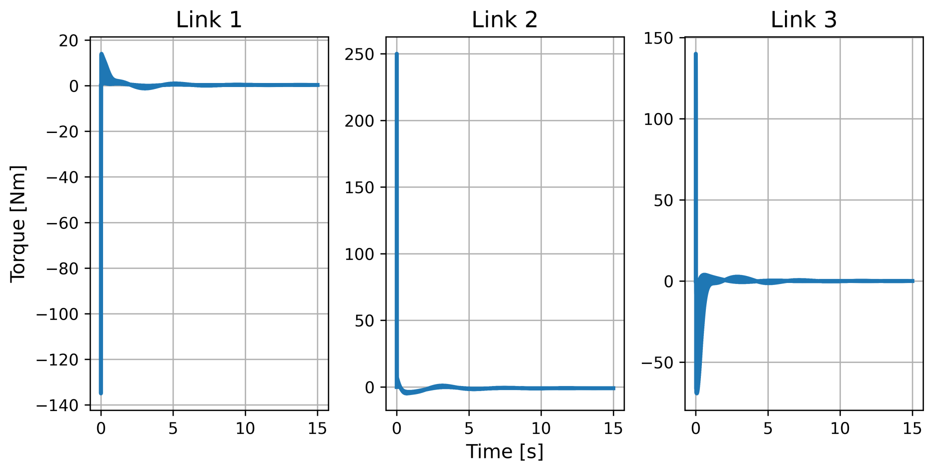

Figure 5.4: RIST - Low-level Joint Torque Trends

velocity were not directly monitored by the controller, they were able to stay relatively stable with little-to-no unnecessary oscillation. The joint states can be seen in figure 5.5.

The remaining analysis for all three tests will revolve around the parameters shown in figure 5.6. These are the ID-QP-specific tracking variables, and the tracking of these four parameters are the primary purpose of this discussion.

The deviation of the COM from the center line, x_c, was able to reach and maintain the desired position, but the height, h_c, and orientation of the COM, q_c, was less consistent. The height differed from equilibrium by approximately $0.0256[m]$, and the angle of link 3, which had the largest error, was $0.1129[rad]$ from the desired position at equilibrium. This is most likely due to the gains given to the ID-QP cost function, which prioritized the

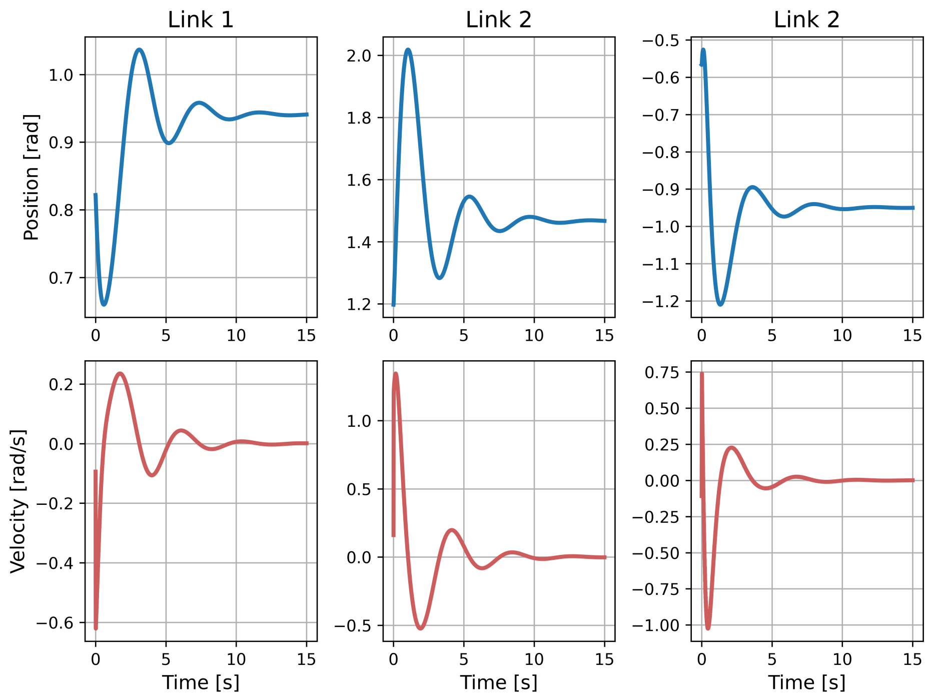

Figure 5.5: RIST - State Trend for TPM During Recovery

tracking of the angular momentum, L_c, and thus indirectly monitors the COM location.

It should be noted that the ID-QP function does not directly track the COM x-position. The MPC algorithm computes the optimal L_c which will stabilize the COM. Then the ID-QP function follows the angular momentum, indirectly giving a high priority to x_c.

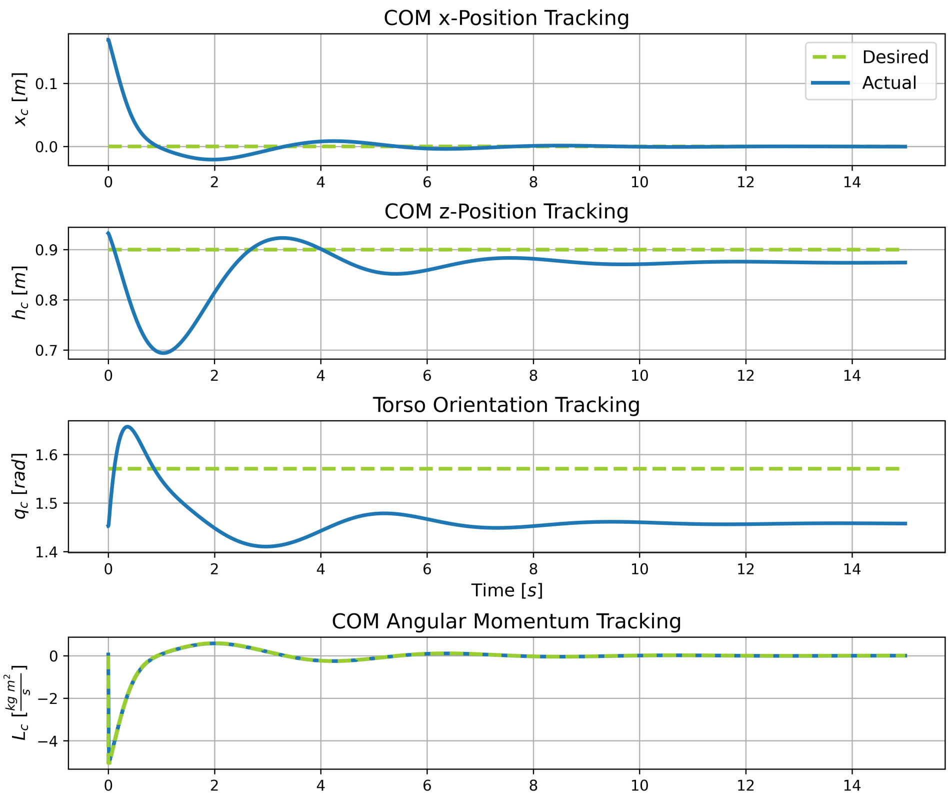

Figure 5.6: RIST - ALIP State Tracking via ID-QP Function

5.2.2 Height Variation Test (HVT)

In this test, the height was adjust mid-simulation in order to evaluate the system's flexibility at differing stance heights. This is especially relevant for when walking is eventually introduced into the system. Gait adjustment during walking is dependent on the accuracy of the COM height as it is used to calculate components like the ZMP and CMP accurately. It should also be mentioned that the HVT is considered a safe test in comparison to the RIST and EDT. For this reason, it was completed second, as its behavior was more telling of whether the gains that were being used in the ID-QP function were appropriate.

Figure 5.7 shows the three tested heights. Starting at $0.90[m]$ before being changed to $0.80[m]$ at $5[s]$ and $1.00[m]$ at $20[s]$. The first transition is during the initial recovery period. This was done to assess the performance when given a new desired position while in motion. The second transition occurs when the system is at a state of equilibrium, but is more dramatic; causing for a considerable amount of overshoot. Figure 5.8 shows the height and error trend for the COM during simulation.

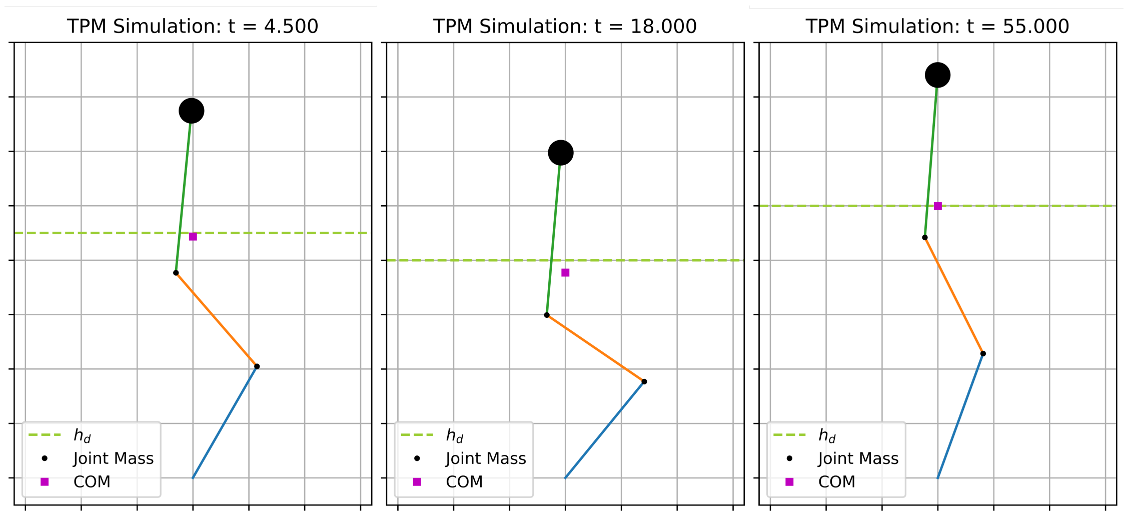
TPM Simulation: t = 4.500
TPM Simulation: t = 18.000
TPM Simulation: t = 55.000
h_d
Joint Mass
COM

Figure 5.7: HVT - Three Primary TPM Positions

The controller is capable of navigating varying COM heights during simulation. The most notable error is seen during the $0.80[m]$ time frame. This is likely because of the configuration of the joints, and their difficulty to position the COM at zero height error without sacrificing the higher priority configuration variables. This is better demonstrated in figure 5.9 where it is shown that the L_c tracking is extremely accurate due to its high priority gain. When the pendulum's COM reaches the zero-position, the desired L_c also goes to zero, making it unlikely that the ID-QP function will correct the orientation error.

Interestingly, the torso orientation error maintained a constant magnitude during the length of the simulation. In figure 5.7 there are black dots at the ends of each of the three TPM links. These represent the masses of each link and their size is related to their magnitude, but they are also slightly misleading. In actuality, the point mass which the

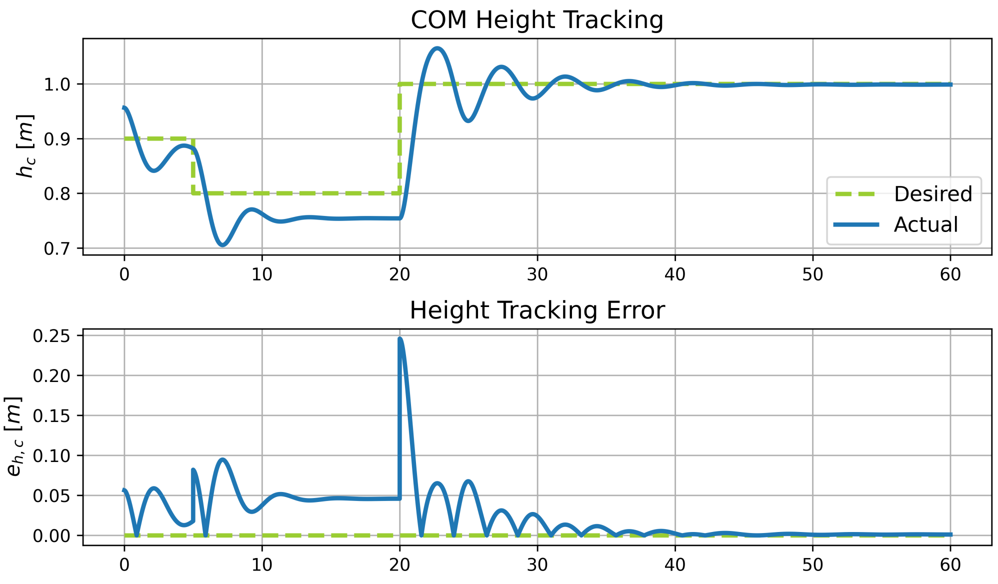

Figure 5.8: HVT - Height Tracking and Error Trend

TPM model uses to represent the joint is located in the middle of each of the links. With this understanding, it is clear that in order to maintain the COM at the zero position, the torso link, which is also the heaviest of the three, must sit as close to the zero position as possible.

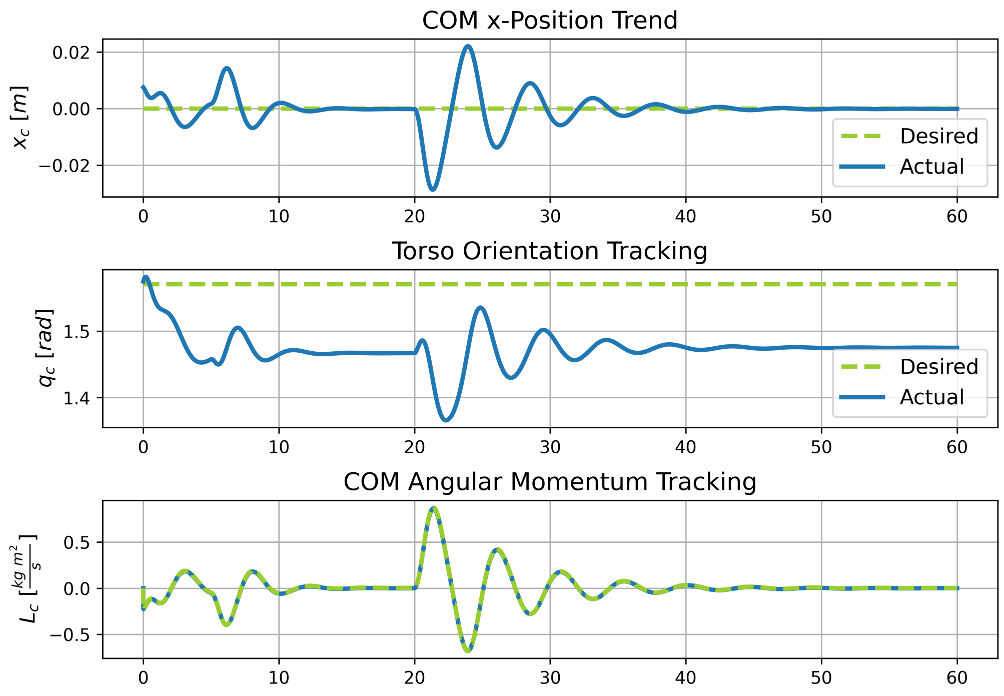

Figure 5.9: HVT - MPC Variable Tracking

5.2.3 External Disturbance Test (EDT)

One of the most important tests a robot must endure is whether or not it can maintain stability when introduced to sudden impacts or pushes. For this reason, the bulk of robot control system design is completed with an emphasis on this characteristic, and the research completed here is no exception. Once the model proved capable of standing at varying stance heights, and recovering from unique initial positions (both relatively safe testing environments), a small function for applying a force to the torso of the MuJoCo simulated TPM was created.

The third and final test performed on the MPC system was an external disturbance test. The pendulum was given a disturbance and observed on its recovery performance, as was the procedure for the RIST and HVT. To be more specific, a $0.1[N]$ disturbance was given to the top-most link of the TPM for $0.01[s]$ while starting from the resting position. Figure 5.10 shows four positions of the pendulum during the simulation.

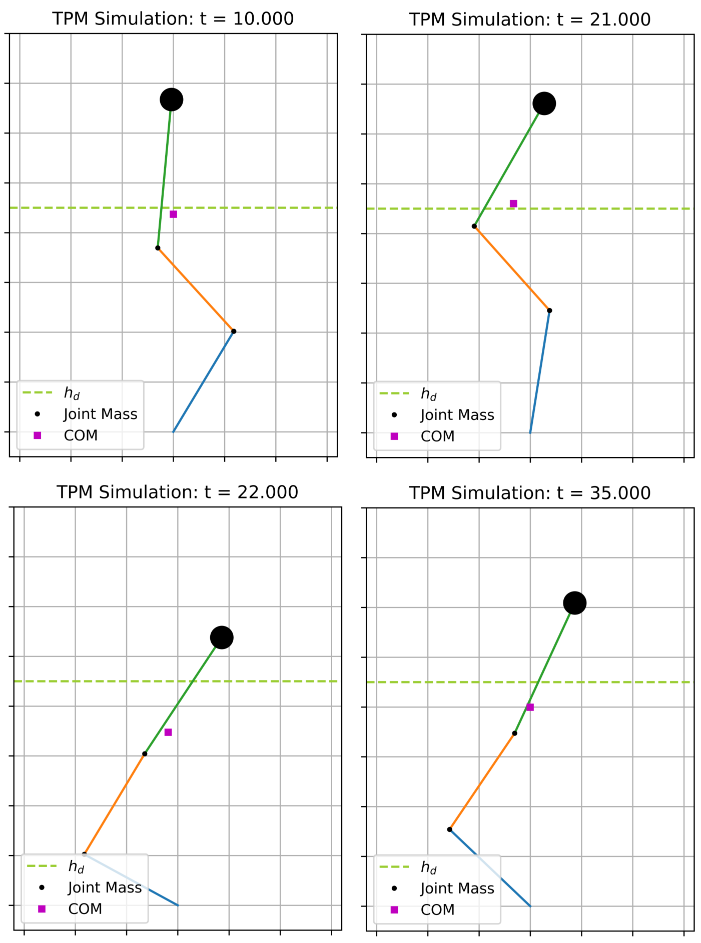

Figure 5.10: EDT - Four Primary TPM Positions

The first position shown is the resting position. The simulation was started and allowed to settle before the disturbance was applied at $20[s]$. The next state is when $t = 21[s]$ and shows the pendulum in motion from the disturbance. The knee joint then stretched out fully and became stuck on the opposite side in image three. After a brief settling period, the pendulum came to rest at a non-ideal orientation. It can be seen that the knee is oriented backwards, and the torso link of the pendulum is askew. The COM is not at the desired height, h_d, but x_c is approximately zero.

Figure 5.11 shows the low-level joint torques directly before, during and after the disturbance was applied. It is recognized that the trend is extremely erratic before leveling off at the equilibrium state. While it is noted that x_c and L_c are tracked very accurately by the controller (figure 5.12), this behavior is very problematic. In an attempt to clamp the magnitude of the pendulum torques, a cost for input was implemented. This may partially explain the trend, as the pendulum may have a hard time recovering from the initial disturbance due to its inability to enact large changes in torque values effectively.

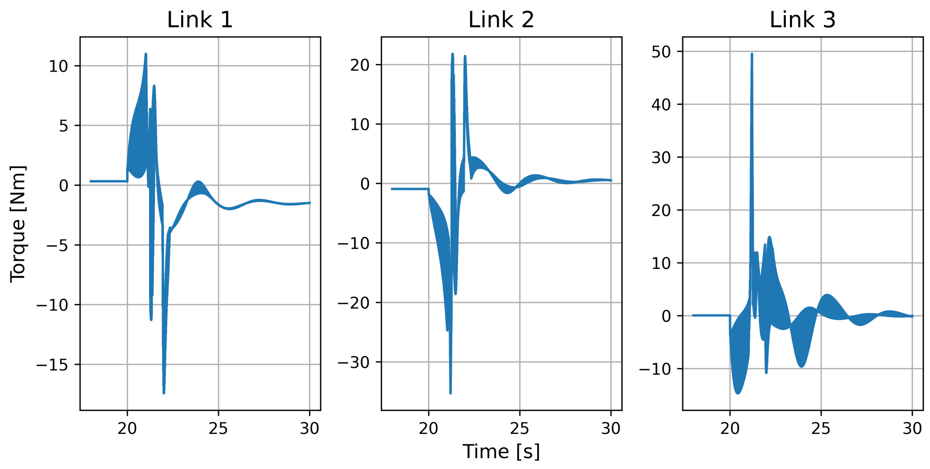

Figure 5.11: EDT - Low-level Joint Torque Trends

Figure 5.12 shows that the error for height and torso orientation tracking settle at a constant error that is larger than those experienced in any previous test. That said, this trend is most likely able to be improved with the proper tuning of the weights in the ID-QP function. Also, the torque cost is currently purely magnitude-based. It may show better results if the cost for torque was dependent on the magnitude change in torque. In this case, the controller would be forced to make smoother torque curves while also being able to reach a stable constant torque trend with zero cost.

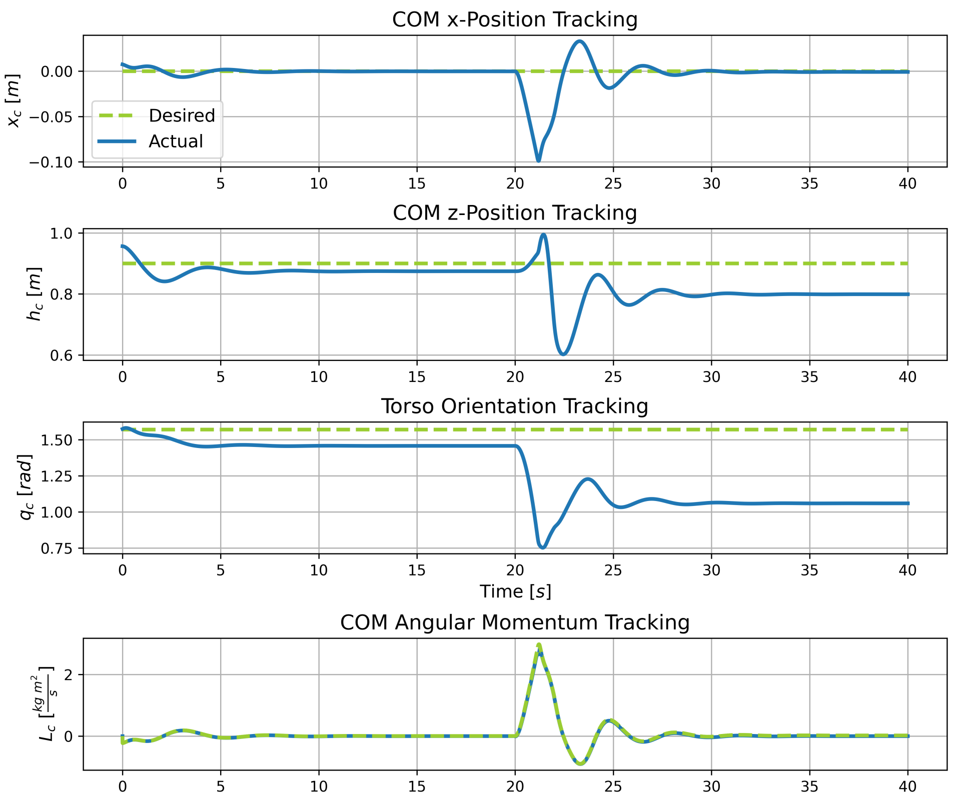

Figure 5.12: EDT - ALIP State Tracking via ID-QP Function

Chapter 6

Conclusions and Future Work

The goal of this project was to present a reliable controller for the triple pendulum model which could safely navigate randomized initial state conditions, variations in height, and external disturbances using the model predictive control formula. A side-objective was also to select an optimization algorithm which could compute the necessary control inputs in an efficient time frame. Overall, these objectives were satisfied, as the controller showed largely acceptable performance.

The controller struggled in the EDT environment, but even then was able to accurately track the L_c and x_c components with a small amount of error. The orientation of the torso link generally maintained a constant degree of error in all tests. Here, the issue can most likely be resolved by re-tuning the ID-QP cost function gains.

As expected, the largest drawback to the MPC system was the run-time necessary for

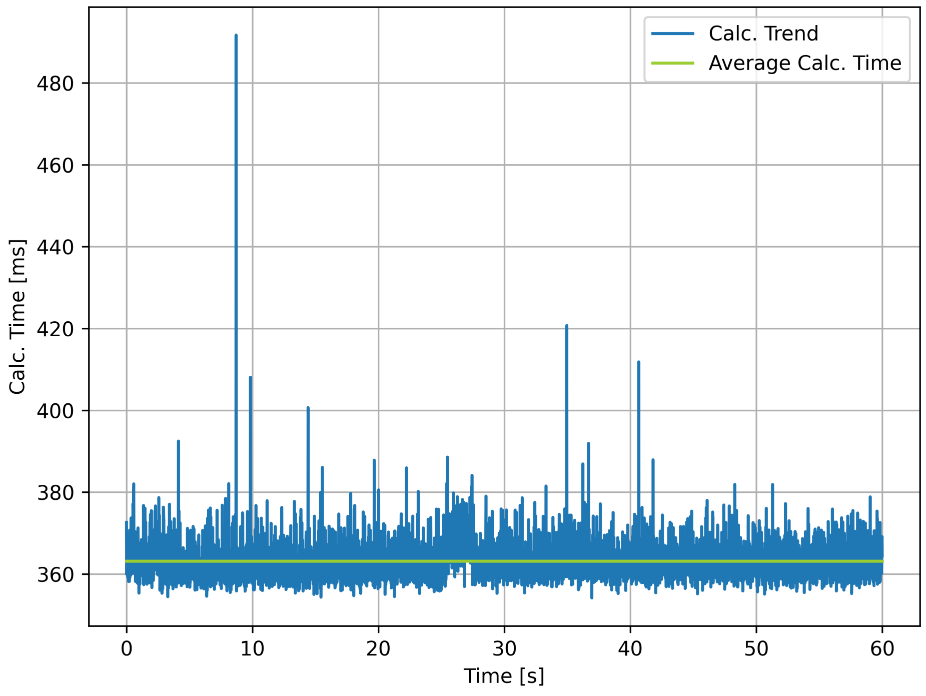

Figure 6.1: MPC Computation Time

each iteration of the search algorithm. The calculation time for an arbitrary isolated test is shown in figure 6.1. The average calculation time was $363.13[ms]$ which was allowable for the purpose of simulation, but is not acceptable for hardware implementation. It was observed that as the PH decreased, the TPM would oscillate about the desired orientation more heavily, increasing the settling time for the controller but greatly improving calculation time. For instance, when the PH was 10 windows (all other parameters set equal), the average calculation time was $61.96[ms]$ which is $5.86x$ faster than the 20 window simulation. That said, it is noted that the system here was programmed in Python and most robotic software systems are built from some form of C++; which runs between 10-100x faster according to online sources.

Continuation of this project will have many avenues. First and foremost, the gains used in the ID-QP function should be expanded to account for joint limits, such that the behavior shown in the EDT is not possible. With proper expertise the ID-QP function can also be reformulated for the Digit model such that the MPC can be tested directly on a 3-D bipedal system.

The strength of the constructed MPC system is that very little, if any, changes would have to be made to incorporate a larger model. Due to the system being linear and the decoupling of the saggital and frontal planes, the COM dynamics and cost function can be easily expanded to 3-D as well.

It would also be worth testing other methods of computing the cost gradient and Hessian, the most computationally expensive function in the entire program. Here, the 2-point central finite difference method was used. Albeit it performed to an acceptable level of accuracy, it is understood that it is a slow and tedious method for calculating derivatives.

Bibliography

[1] Numchai Narkvitul, Prapart Ukakimaparn, and Thanit Trisuwannawat. "Closed-form formulas for continuous/discrete-time PID controllers' parameters". In: *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*. ISSN: 2093-7121. Oct. 2014, pp. 1526–1530. DOI: 10.1109/ICCAS.2014.6987808.

[2] Ravi Kumar Mandava and Pandu R. Vundavilli. "An optimal PID controller for a biped robot walking on flat terrain using MCIWO algorithms". en. In: *Evol. Intel.* 12.1 (Mar. 2019), pp. 33–48. ISSN: 1864-5909, 1864-5917. DOI: 10.1007/s12065-018-0184-y. URL: http://link.springer.com/10.1007/s12065-018-0184-y (visited on 05/13/2022).

[3] Chao Zhang et al. "System Design and Balance Control of a Bipedal Leg-wheeled Robot". In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dec. 2019, pp. 1869–1874. DOI: 10.1109/ROBIO49542.2019.8961814.

[4] Guillermo A. Castillo et al. "Reinforcement Learning Meets Hybrid Zero Dynamics: A Case Study for RABBIT". en. In: *arXiv:1810.01977 [cs]* (Oct. 2018). arXiv: 1810.01977. URL: http://arxiv.org/abs/1810.01977 (visited on 03/02/2022).

[5] Guillermo A. Castillo et al. "Hybrid Zero Dynamics Inspired Feedback Control Policy Design for 3D Bipedal Locomotion using Reinforcement Learning". en. In: *arXiv:1910.01748 [cs]* (Oct. 2019). arXiv: 1910.01748. URL: http://arxiv.org/abs/1910.01748 (visited on 03/02/2022).

[6] Richard Beranek, Masoud Karimi, and Mojtaba Ahmadi. "A Behavior-Based Reinforcement Learning Approach to Control Walking Bipedal Robots Under Unknown Disturbances". In: *IEEE/ASME Transactions on Mechatronics* (2021). Conference Name: IEEE/ASME Transactions on Mechatronics, pp. 1–11. ISSN: 1941-014X. DOI: 10.1109/TMECH.2021.3120628.

[7] He Li, Robert J. Frei, and Patrick M. Wensing. “Model Hierarchy Predictive Control of Robotic Systems”. en. In: *IEEE Robot. Autom. Lett.* 6.2 (Apr. 2021). arXiv: 2010.08881, pp. 3373–3380. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3061322. URL: http://arxiv.org/abs/2010.08881 (visited on 03/02/2022).

[8] Milad Shafiee-Ashtiani et al. “Push recovery of a humanoid robot based on model predictive control and capture point”. In: *2016 4th International Conference on Robotics and Mechatronics (ICROM)*. Oct. 2016, pp. 433–438. DOI: 10.1109/ICRoM.2016.7886777.

[9] Zohaib Aftab, Thomas Robert, and Pierre-Brice Wieber. “Ankle, hip and stepping strategies for humanoid balance recovery with a single Model Predictive Control scheme”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. ISSN: 2164-0580. Nov. 2012, pp. 159–164. DOI: 10.1109/HUMANOIDS.2012.6651514.

[10] Yang Wang and Stephen Boyd. “Fast Model Predictive Control Using Online Optimization”. In: *IEEE Transactions on Control Systems Technology* 18.2 (Mar. 2010). Conference Name: IEEE Transactions on Control Systems Technology, pp. 267–278. ISSN: 1558-0865. DOI: 10.1109/TCST.2009.2017934.

[11] Neha Raghu. “Model Predictive Control: History and Development”. en. In: *International Journal of Engineering Trends and Technology* 4.6 (2013), p. 3.

[12] Avik Jain et al. *Optimal Cost Design for Model Predictive Control*. en. arXiv:2104.11353 [cs, eess]. June 2021. URL: http://arxiv.org/abs/2104.11353 (visited on 06/25/2022).

[13] MASANAO Aoki. “On Optimal and Suboptimal Policies in Control Systems”. en. In: *Advances in Control Systems*. Ed. by C. T. Leondes. Vol. 1. Elsevier, Jan. 1964, pp. 1–53. DOI: 10.1016/B978-1-4831-6717-6.50006-1. URL: https://www.sciencedirect.com/science/article/pii/B9781483167176500061 (visited on 06/25/2022).

[14] Moritz Diehl. “Optimization Algorithms for Model Predictive Control”. en. In: *Encyclopedia of Systems and Control*. Ed. by John Baillieul and Tariq Samad. London: Springer London, 2013, pp. 1–11. ISBN: 978-1-4471-5102-9. DOI: 10.1007/978-1-4471-5102-9_9-1. URL: http://link.springer.com/10.1007/978-1-4471-5102-9_9-1 (visited on 03/02/2022).

[15] B. T. Polyak. “Newton’s method and its use in optimization”. en. In: *European Journal of Operational Research* 181.3 (Sept. 2007), pp. 1086–1096. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2005.06.076. URL: https://www.sciencedirect.com/science/article/pii/S0377221706001469 (visited on 05/19/2022).

[16] S. Kajita et al. “The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation”. In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 1. Oct. 2001, 239–246 vol.1. DOI: 10.1109/IROS.2001.973365.

[17] Yukai Gong and Jessy Grizzle. “Zero Dynamics, Pendulum Models, and Angular Momentum in Feedback Control of Bipedal Locomotion”. en. In: *arXiv:2105.08170 [cs, eess]* (Feb. 2022). arXiv: 2105.08170. URL: http://arxiv.org/abs/2105.08170 (visited on 03/02/2022).

[18] Predrag S. Stanimirović and Marko B. Miladinović. “Accelerated gradient descent methods with line search”. en. In: *Numer Algor* 54.4 (Aug. 2010), pp. 503–520. ISSN: 1572-9265. DOI: 10.1007/s11075-009-9350-8. URL: https://doi.org/10.1007/s11075-009-9350-8 (visited on 05/29/2022).

[19] Jerry Pratt et al. “Capture Point: A Step toward Humanoid Push Recovery”. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*. ISSN: 2164-0580. Dec. 2006, pp. 200–207. DOI: 10.1109/ICHR.2006.321385.

[20] Hyobin Jeong et al. “Biped walking stabilization based on foot placement control using capture point feedback”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Sept. 2017, pp. 5263–5269. DOI: 10.1109/IROS.2017.8206418.

[21] Jenna Reher, Claudia Kann, and Aaron D. Ames. “An Inverse Dynamics Approach to Control Lyapunov Functions”. In: *2020 American Control Conference (ACC)*. ISSN: 2378-5861. July 2020, pp. 2444–2451. DOI: 10.23919/ACC45564.2020.9147342.