UNIVERSITY OF CALIFORNIA

Los Angeles

Real-Time Optimization for Control of a Multi-Modal Legged Robotic System

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Mechanical Engineering

by

Joshua Rosenberg Hooks

2019

ABSTRACT OF THE DISSERTATION

Real-Time Optimization for Control of a Multi-Modal Legged Robotic System

by

Joshua Rosenberg Hooks

Doctor of Philosophy in Mechanical Engineering

University of California, Los Angeles, 2019

Professor Dennis Hong, Chair

For decades humans have been trying to create machines that can mimic the capabilities of legged animals. Already humans have developed cars capable of traveling long distances at high speeds, and automation robots able to perform incredibly accurate tasks at high rates. However, the machines and robots that we have created so far only operate in specific controlled environments. What is impressive about animals is their capability of navigating a multitude of environments. Further still, humans and select animals are not only able to navigate the world but they are also capable of manipulating the world around them. It is these abilities that we wish to replicate in robots in order to create machines that can aid humans in their everyday lives.

In recent years, there has been a large surge in the number of quadrupedal robots available for commercial use. They have shown that they can navigate the world at speeds comparable to human walking or jogging. However, as of now no quadrupedal robot has the capability to help with tasks other than search or surveillance type of applications. Humanoid robots have also made large advancements in the past decade. Multiple full sized humanoids have demonstrated stable walking and locomotion. However, the most important development for humanoid robots is their ability to manipulate their environment. The DARPA Robotics Challenge show cased humanoid robots driving cars, opening doors, and using power tools to complete tasks. Unfortunately, right now there are no commercially available humanoid robots due in most part to their lack of safety and reliability. Unlike quadrupedal robots, humanoids are not statically stable and thus require active balancing at all times. This added complexity makes humanoid robots dangerous to operate in human environments.

This dissertation presents a novel quadrupedal robot and control strategy that brings legged robotics closer to one day aiding humans in their everyday life. The novel quadrupedal robot, named ALPHRED, uses a non-traditional kinematic structure to provide multiple modes of operation, giving the robot capabilities beyond that of traditional quadrupeds. The online path planner was developed using a non-linear program to solve for viable center of mass trajectories and footstep locations based off of the dynamics of the robot and height map information from vision data. In addition, a model predictive controller was developed to track the desired motion of the robot. This model predictive controller uses massless leg dynamics to predict the dynamics of the body but also incorporates known swing leg dynamics in a novel fashion that results in a 250 Hz motion tracking controller. Using the developed control schemes the robot is capable of navigating uneven terrain and can achieve a top speed of 1.5 m/s on flat terrain. ALPHRED is equipped with passive wheels on it's belly that allow it to push itself around on flat ground for an efficient and fast form of locomotion. Finally, ALPHRED is capable of balancing on two limbs freeing up the other two limbs for manipulation tasks. Using all of the mode's of operation ALPHRED is the first quadrupedal robot that is capable of picking up varying size packages and completing an end-to-end package delivery.

The dissertation of Joshua Rosenberg Hooks is approved.

Robert M'Closkey

Veronica Santos

Tsu- Chin Tsao

Dennis Hong, Committee Chair

University of California, Los Angeles

2019

To my family and friends.

Table of Contents

List of Figures

Ackowledgments

Firstly, I would like to thank my advisor, Dr. Dennis Hong, for all of his support through out the past five years. His excitement, positive outlook, and overall bigger than life personality is a rarity in this world and was a major factor in getting me through the many highs and lows of a PhD. His encouragement to get your hands dirty, to break the robots, and to push the boundaries all in order to learn has forever changed the way I approach difficult challenges, and for that I am eternally grateful. When I first walked into Dr. Hong's office he had just moved to UCLA and his lab, RoMeLa, was completely empty. I am proud to say that now five years later the lab is overflowing with equipment and robots of all different shapes and sizes. It is one of the greatest things that I have ever been apart of, and I am very thankful to Dr. Hong for allowing me to be apart of the team to help him build his lab at UCLA.

I am grateful to have a committee that I respect as researchers, teachers, and people. It is not a coincidence that I chose professors that are not only renowned in their fields but who also care about the students that they teach. Dr. Robert M'Closkey, Dr. Veronica Santos, and Dr. Tsu-Chin Tsao thank you for always having time for my questions and guiding me when the time arose.

During my studies at UCLA I was fortunate to be a part of one of the best robotics labs in the world. Not only was I able to interact with some of the smartest people in the field of robotics but they were also some of the nicest. I want to say thanks to everyone who was a part of RoMeLa for helping me both with my academic career but also for being there for me as a friend. Specifically, I would like to thank Jeffrey Yu, Alexie Pogue, Min Sung Ahn, and Xiaoguang Zhang who were there from day one. I am sure that I could not have made it through without all of your support. I would also like to give a special thanks to Taoyuanmin Zhu. His knowledge of electro-mechanical systems might be unparalleled but even more importantly his willingness to always help is what I am truly thankful for. I am not sure I can count the amount of times Taoyuanmin helped fix a communication issue, manufactured a quick fix for a broken chassis, or just pointed out a negative sign that should not have been there.

I would also like to give a special thanks to Julian Zhou, Yi Zheng, and Martin Lee for getting me through my first year of graduate school. For anyone who has gone through a PhD program they know that passing the PhD preliminary exam is no small feet and I am not sure I could have done it without the help of these three.

Finally and maybe most importantly I would like to thank my friends and family. There has never been a moment in my life where I did not have the love and support of my family. There are some very hard times in a PhD where it is often hard to see the light at the end of the tunnel and I just want to say thanks to my family for always showing me the way. I have a unique group of friends who are very successful in their own respective careers. They inspired me to continue in my passion but beyond that they have always been there for me whether through hardships related to my PhD career or just life hardships in general. I just wanted to say thank you for getting me to the end.

Curriculum Vitae

2008 – 2012	B.S. in Mechanical Engineering, University of Washington
2012 – 2014	Controls engineer at Electroimpact, automatic fiber placement division
2014 – Present	Ph.D. student in Mechanical and Aeronautical Engineering, University of California, Los Angeles (UCLA).

Publications

Dennis Hong, Joshua Hooks, Jeffrey Yu, and Min Sung Ahn. Encyclopedia of robotics. In *Design of Legged Robotics*. Springer, 2019.

Joshua Hooks, , Min Sung Ahn, Jeffrey Yu, Xiaoguang Zhang, Taoyuanmin Zhu, and Dennis Hong. Alphred: A multi-modal operations quadruped robot for package delivery applications. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

Joshua Hooks, Min Sung Ahn, and Dennis Hong. Online trajectory optimization for legged robotics incorporating vision for dynamically efficient and safe footstep locations. In *2019 ASME Intrnational Design Engineering Technical Conference and Computers and Information in Engineering Conference (IDETC-CIE)*, 2019.

Joshua Hooks and Dennis Hong. Implementation of a versatile 3d zmp trajectory optimization algorithm on a multi-modal legged robotic platform. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3777–3782. IEEE, 2018.

Jeffrey Yu, Joshua Hooks, Sepehr Ghassemi, and Dennis Hong. Exploration of turning strategies for an unconventional non-anthropomorphic bipedal robot. In *ASME 2017 International Design*

Engineering Technical Conferences and Computers and Information in Engineering Conference, pages V05BT08A021–V05BT08A021. American Society of Mechanical Engineers, 2017.

Jeffrey Yu, Joshua Hooks, Sepehr Ghassemi, Alexandra Pogue, and Dennis Hong. Investigation of a non-anthropomorphic bipedal robot with stability, agility, and simplicity. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2016 13th International Conference on*, pages 11–15. IEEE, 2016.

Jeffrey Yu, Joshua Hooks, Xiaoguang Zhang, Min Sung Ahn, and Dennis Hong. A proprioceptive, force-controlled, non-anthropomorphic biped for dynamic locomotion. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

Taoyuanmin Zhu, Joshua Hooks, and Dennis Hong. Design, modeling, and analysis of a liquid cooled proprioceptive actuator for legged robots. In *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2019.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Robotics have had a large impact on this planet. For decades we have seen robots manufacture our cars, package our food, and build our electronics. But we have not seen what science fiction has promised us, and that is robot's aiding us in our day to day lives. We have seen robots have incredible success in factories and automation but not in the real world. This is due in large part to the fact that the approaches to effectively navigating the two different environments is extremely different. In factories, the environment is controlled and the state of the robot can be measured to a high degree of accuracy. Additionally, the robots are typically fully actuated giving them full control authority at all times. With this framework, brushless DC (BLDC) motors with large gear reductions are the preferred method of actuation because they provide both strength and precision. These fully actuated systems can use simple control strategies such as high gain PD joint level controllers or inverse dynamic whole-body controllers [74], [47], [17].

Unfortunately, this is not the case with legged robots. By removing the fixed contact to the ground the system becomes under-actuated, which limits the control authority of the robot at a given point of time [73]. Techniques like inverse dynamics are no longer feasible and instead trajectories must be planned over finite time horizons in order to account for the complex dynamics required to complete high level goals. The state of the robot is now no longer able to be directly measured, but instead an array of sensor measurements and advanced filtering techniques are required. In addition, the control of the robot is now done through controlling the ground reaction forces at changing contact points. This poses two great challenges. First, the problem is now no longer continuous; as the robot travels different feet are in contact with the ground at different times, each different combination of contacts results in a different dynamical model [3], [33]. Second, when a foot makes contact with the ground a collision occurs imparting an impulse onto the system. These collisions must be effectively mitigated and controlled in order to provide stable locomotion. The actuators used for robots in factories are not capable of handling impulses and this has required new types of actuators that incorporate compliance [29].

In the past couple of decades the robotics community has come to the realization that the mechanical designs and control techniques used for automation robots are not adequate for legged robots. The traditional type of actuation used in automation robotics consists of a BLDC motor with a large gear reduction transmission. This type of actuation provides large torques with high precision but is not capable of handling large impulses. Large gear reductions not only amplify the torque in the system but they also amplify the inertia and friction of the system. The amplification of these two parameters causes the transmission to lockup, when being back driven, changing the dynamics of the system, and can often lead to the failure of the actuator. There have been three main types of actuators developed for legged robotics to address the short comings of the traditional actuators used for automation robots.

The first is hydraulic actuators, which function by controlling the pressure difference in hydraulic fluid lines through a system of compressors and valves. These type of actuators are extremely powerful and traditionally used on large construction equipment where weight and precision are not a concern. However, Boston Dynamics [1] showed that these types of actuators can be extremely effective for legged robotics with their implementation on Atlas and Big Dog [66]. Second, are series elastic actuators (SEA) which insert an elastic element in between the load and the actuator itself [64]. Accurate torque control is achieved by measuring and controlling the deflection of the compliant element. There have been a number of successful robotic platforms that use SEAs [39, 55, 69, 34, 51]. Lastly, are proprioceptive actuators or quasi-direct drive (QDD) actuators that use a low gear reduction transmission combined with a high-torque large air-gap radius BLDC. Sangbae Kim and Daniel Koditschek were the first to discuss how the transmission transparency of proprioceptive actuators are ideal for legged robotic applications [71, 50]. Since then there have been a number of robotic platforms that have utilized this type of actuator [12, 48, 11, 62, 2]

The control of legged robotics has also seen large developments beyond the techniques used for fully actuated systems. Some of the first successful control strategies were developed by the MIT Leg Laboratory under Mark Raibert. The Raibert controllers used powerful heuristics to control the velocity and orientation of the robot in task space rather than control each joint individually in joint space [67]. Kajita et al. was one of the first to model the dynamics of legged robots as an inverted pendulum [44]. Through this method he developed the zero moment point (ZMP) stability criteria; this condition is satisfied when the center of pressure (CoP) exists inside the support polygon of the robot. When this condition is met the dynamics of the problem become fully actuated allowing for analytical solutions to the problem. Using this criteria Kajita was able to control a bipedal robot through the analytical solution of an LQR [43]. The success of the ZMP controller lead to many different variations of ZMP control [76], and the development of the capture point (CP) method [65], which used the linear inverted pendulum model to determine the footstep location that would drive the pendulum model to the unstable equilibrium point. This method was first used as a push recovery method but was then further expanded as a walking controller [22]. These analytical solutions proved to be tools for controlling legged robots however they do not naturally extend to dynamic locomotion with periods of flight, nor are they capable of handling additional constraints such as friction.

To produce richer solutions the robotics community turned to numerical solutions using optimization techniques. Trajectory planners use simplified template models of the robot dynamics, such as a linear inverted pendulum, while including additional constraints to plan for viable CoM trajectories and footstep plans over finite horizons [87, 86, 88, 18, 57]. Whereas, motion controllers now use the full dynamics of the system to produce whole-body control of the robots [83, 6, 70]. While these whole-body controllers produce impressive whole-body solutions they only consider the current state of the robot and do not account for future events. As noted before, legged locomotion consists of known discrete transitions from footstep to footstep. To account for these predictable transitions model predictive control (MPC) has been utilized to optimize for trajectories and ground reaction forces over a finite horizon. With an MPC only the first solution is used before the problem is re-formulated and re-planned. Due to the constant re-planning, MPC's are quite computationally heavy and haven't been viable for real-time control on real-world robotic platforms until the past couple of years [52, 19].

While the world has seen large advancements in the field of legged robotics we still have yet to see legged robots be utilized to aid humans in any industry. This dissertation aims at combining the advancements in both the mechanical design of legged robots and the control of legged robots to produce a robot that is one step closer of achieving the ultimate goal of robots helping humans. The robot presented in this work, uses propriocpetive actuators integrated into a novel kinematic structure to produce a robot that is capable of multiple modes of operation for various different tasks. By combining a cutting edge MPC controller with the novel mechanical design of the robot, the robot was able to pickup a package and place it on it's back, walk to a delivery location, and finally grab the object from its back and drop of the package at the desired location. To the best of this authors knowledge this is the first quadrupedal robot that is capable of end-to-end package delivery.

1.2 Background

1.2.1 Trajectory Optimization

As the control of legged robotics has evolved the reliance on numerical solutions has steadily increased, making optimization a vital tool for both trajectory planners and motion tracking controllers. A general optimization problem is the numerical solution to a problem that minimizes (or maximizes) some measure of performance while adhering to a multitude of constraints. A trajectory optimization (TO) problem solves for a locally optimal trajectory given a linear or non-linear dynamical system, $\dot{x}(t) = f(t, x, u)$. The solution to a TO problem will provide the state $x(t)$ and control $u(t)$ over the entire trajectory. To illustrate a TO problem, imagine a bipedal robot walking across a room while avoiding a table in the middle of the room. The path that the robot takes would be the *trajectory* that it follows whereas the torques in the actuators of the robot to create that trajectory would be the control. A trajectory is said to be *feasible* if it adheres to all of the constraints. The constraints in this example would be the system dynamics, collision avoidance, boundary conditions (i.e. reaching the final goal position), etc. The *optimal* trajectory is selected by choosing the best of all the feasible trajectories. In this example best could mean the trajectory that takes the shortest amount of time or the trajectory that uses the least amount of energy. Trajectory optimization is used when the problem has no closed-form solution or the solution is too hard or impractical to calculate analytically [49].

There are many ways to formulate a TO problem however, for the purposes of this paper we will restrict our definition to a differentiable continuous-time dynamical system. The typical TO problem tries to determine the "best" trajectory by minimizing a cost function (1.1) by adjusting the problems *decision variables* in order to find the optimal solution. Common costs for trajectory optimization for legged robots are total torque, total time, total acceleration, etc.

$$\min_{x(t),u(t)} J(t, x(t), u(t)) \quad \text{Cost function} \tag{1.1}$$

The minimization of the cost function is subject to a variety of different constraints (1.2) - (1.5), the first being the dynamics of the system. This constraint describes how the system evolves through time based off of the control effort. In many cases the system dynamics will be drastically simplified (template model) in order to decrease the solve time of the TO problem.

$$\dot{x}(t) = f(t, x(t), u(t)) \quad \text{system dynamics} \tag{1.2}$$

Next, are the path constraints. This type of constraint encompasses a wide variety of constraints that restrict the trajectory. An example of this could be to prevent the robot from colliding with an object or prevent the robot from taking a step that is beyond it's kinematic reachability.

$$h(t, x(t), u(t) \leq 0 \quad \text{path constraints} \tag{1.3}$$

The boundary constraints are to ensure that the robot starts and ends in the right conditions.

$$g(t_0, t_F, x(t_0), x(t_F)) \quad \text{boundary constraints} \tag{1.4}$$

Finally, both the state and the control effort can be bounded from above and below. For example, the control effort could be bounded by the known torque limits of a robot's actuators.

$$x_{lower} \leq x(t) \leq x_{upper} \quad \text{state bounds} \tag{1.5}$$

$$u_{lower} \leq u(t) \leq u_{upper} \quad \text{control bounds} \tag{1.6}$$

1.2.1.1 Problem Formulations

There are a variety of ways to formulate a trajectory optimization problem [49, 8, 9]. The first two major methods are the *direct* method and the *indirect* method [79]. The difference between the two methods is the selection of decision variables and how the dynamics of the problem are incorporated. The indirect method uses only control inputs as decision variables. If both the initial state and the control inputs are known then a forward propagation of the dynamics can be performed to calculate the entire trajectory of the problem. Whereas, with a direct method the state and control are both decision variables. This method results in a larger number of decision variables however the problem is now broken up into multiple optimization problems. With the indirect method a small change to a state at the beginning will have a large effect on the states at the end, this is not the case in the direct method. The performance of the two methods is highly dependent on the type of problem that is being solved.

Often times the optimization problem can be broken up into a set of smaller optimization problems, this process is known as *transcription*. Transcription breaks up the continuous time dynamics of the problem into a finite set of nodes or *knot points*. The dynamics can be propagated forward explicitly using something like Runga-Kutta, this method is known as the *shooting* method. Alternatively, the continuous time dynamics can be approximated by polynomial splines in between the knot points, which is known as *collocation* (fig. 1.1). Constraints are added to the problem formulation to ensure continuity between two adjacent polynomials. For this formulation the decision variables consist of the coefficients of the polynomials and the control to create these polynomials [60] [61]. Increasing the number of knot points will increase the accuracy of the approximation, but it will also increase the computation time. Accuracy can also be improved by using a higher order polynomial spline in-between knot points, but again this increases the number of decision variables which increases the computation time.

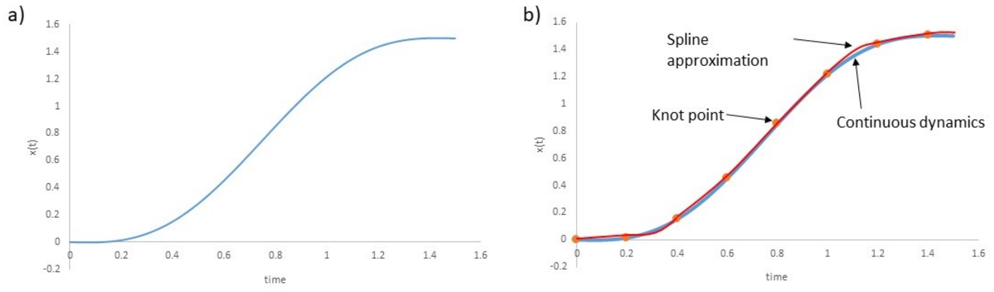


Figure 1.1: a) Continuous time trajectory. b) Continuous time trajectory discretized into a finite number of knot points with polynomial approximations between points.

1.2.1.2 Non-Linear Programming

The tools and techniques to solve a TO problem depend on the formulation of the problem. Depending on the structure of the cost function and constraints allows for different techniques to be used. The most general formulation is a Non-Linear Program (NLP) which is constructed as shown in (1.7) where the cost function, equality constraints, and inequalities constraints can all be non-linear. NLP's can be solved using commercial NLP solvers like

IPOPT [81] or SNOPT [30]. NLP's often times do not find the global minimum but usually result in a local minimia of the problem. Additionally, NLP's are not guaranteed to converge therefore it is imparative that the problem is conditioned appropriately to be used for TO problems on robotic systems.

$$\begin{aligned}
\min_{z} \quad & J(z) \\
\text{s.t.} \quad & \\
& f_i(z) \leq 0 \qquad i = 1, ..., m \\
& h_i(z) = 0 \qquad i = 1, ..., p
\end{aligned} \tag{1.7}$$

Where z is the set of decision variables, $J(z)$ is the cost function, $f_i(z)$ is the i^{th} inequality constraint, and $h_i(z)$ is the i^{th} equality constraint.

1.2.1.3 Convex Optimization and Quadratic Programming

To decrease the computation time of a TO problem, mathematical simplifications can be made to the system dynamics and constraints to turn the problem into a convex optimization problem. A convex optimization problem requires that the cost function in (1.7) is a convex function, the inequality constraints are convex functions, and the equality constraints are affine functions. A convex problem can typically be solved much more efficiently than a NLP with not only a gaurantee on convergence but a gaurantee to converge on the globally optimal solution. Commercially available software like CVX [31] is readily available to solve Convex problems.

Solve times can be further reduced if the problem can be simplified and formulated into a Quadratic Program (QP). QP's are a specific type of Convex program that takes the form of (1.8) where the cost is quadratic and the inequalities are affine. If the problem can be formulated into a QP there are a multitude of commercial solvers such as Gurobi [59], OSQP [75], and qpOASES [27].

$$\begin{aligned} \min_{z} \quad & 1/2 z^T M z + c^T z \\ \text{s.t.} \quad & \\ & Az \leq b \end{aligned} \tag{1.8}$$

Any additional structure to the optimization problem can help to reduce the time it takes to solve the problem. If the cost or constraints consist of upper triangular, block diagonal, or strictly diagonal matrices sparse matrix libraries can be used to solve the problem faster. Using an initial guess that is close to the optimal solution will also help to reduce solve times. This is typically applicable to TO problems in robotics where the solution to the previous problem is used as the initial guess to the next problem, this method is called a *warm start* to the problem [82].

1.2.2 Modeling Dynamics

A crucial element to the success of an optimization problem for legged robotics is the ability to model the dynamics of the system. The system can be modeled as accurately as possible taking into account all of the non-linearities such as friction in the joints and the compliance of the links, however this is typically very challenging to do and results in optimization problems that take minutes or even hours to solve. Often times simplifications are made to the model to provide for tractable optimization problems, these simplified models are called template models. The challenge for a controls engineer is to capture enough of the actual dynamics in the template model to successfully control the robot.

1.2.2.1 Rigid Body Dynamics

The way in which a rigid body moves through space and interacts with other rigid bodies is a well known field in the area of dynamics. Typically a legged robotic system uses a floating base model meaning that the body of the robot is represented by a spatial position vector q_B consisting of six states, three rotational and three linear [23, 24]. Each one of the robots links is treated as a separate rigid body connected to each other by a single joint with position q_i. The vector $\boldsymbol{q} = [q_B^T, q_1, q_2, ...q_{n-6}]^T$ represents the current position of the robot base and all of the joints. From this representation the dynamics of the system can be represented by (1.9) [72].

$$\boldsymbol{H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = J(q)^T F + S\tau} \tag{1.9}$$

Where $H \in \mathbb{R}^{n\times n}$ is the mass matrix, $C \in \mathbb{R}^{n\times n}$ is the Coriolis matrix, and $G \in \mathbb{R}^n$ is the gravitational term. $J \in \mathbb{R}^{6N\times n}$ and $F \in \mathbb{R}^{6N}$ is the combined support Jacobians and the collected ground reaction forces respectively for N number of feet. The matrix $S = [\mathbf{0}_{(n-6)\times 6}\mathbf{1}_{(n-6)\times(n-6)}]$ is a selection matrix that only picks off the joints that are actuated. For a complete discussion on rigid body dynamics and efficient algorithms for computing the above matrices please see [25].

Although the full rigid body dynamics equations in (1.9) are quite effective at representing the dynamics of a real-world system they can often times be too burdensome to use due to their computational load. Many different simplifications can be made to improve the efficiency of the dynamics but the most common in legged robotics is to assume that the legs of the robot are massless. If the legs have no mass then they can not impart any moments or forces onto the body of the robot and therefore they can be removed from the equations of motion. After applying the massless leg assumption the equation of motion simplify down to:

$$\ddot{\boldsymbol{r}} = \left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{4}} \boldsymbol{f_i}/m \right) - \boldsymbol{g} \tag{1.10}$$

$$\frac{d}{dt}(\boldsymbol{I\omega}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{4}} \boldsymbol{d_i} \times \boldsymbol{f_i} \tag{1.11}$$

where $r \in \mathbb{R}^3$ is the linear position of the body, $f_i \in \mathbb{R}^3$ is the ground rection forces of foot i, $I \in \mathbb{R}^{3\times3}$ is the inertia tensor of the robot, $\omega \in \mathbb{R}^3$ is the angular velocity of the body, and $d_i \in \mathbb{R}^3$ is the vector from the CoM to foot i. While this is a very large simplification it has been proven to be effective if the ratio of the mass of the body to the mass of a limb of the robot is sufficiently large.

1.2.2.2 Zero Moment Point

Continuing with the massless leg assumption the dynamics of the robot can be simplified further. By modeling the system as a linear inverted pendulum (LIP) the equations of motion can be written in a linear equation. In [43] they showed that the equations of motion for this model can be derived from either a 3D linear inverted pendulum or equivalently from a cart pole model as shown in fig. 1.2. This model can be used to control the zero moment point (ZMP) which has been shown to be able to model gaits of legged systems [80]. It was found that if the ZMP is contained within the support polygon of the system, the convex hull that is created from all points in contact with the ground, then the system is stable.

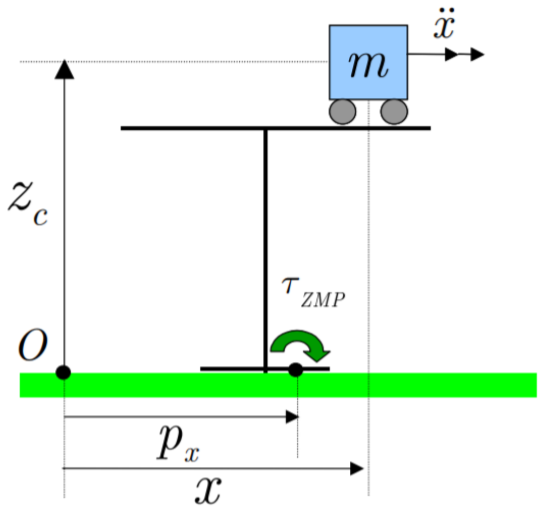


Figure 1.2: Cart pole model, image taken from [43]

Solving for the system dynamics of the simplified model in the x direction yields:

$$\tau_{zmp} = mg(x - p_x) - m\ddot{x}z_c = 0 \tag{1.12}$$

Where x is the x position of the cart, p_x is the ZMP, m is the mass of the cart, and z_c is the constant height of the cart. A similar equation can be derived for the the y direction.

$$\tau_{zmp} = mg(y - p_y) - m\ddot{y}z_c = 0 \tag{1.13}$$

Rearranging these equations we get the famous linear ZMP dynamics equations.

$$\ddot{x} = \frac{g}{z_c}(x - p_x) \tag{1.14}$$

$$\ddot{y} = \frac{g}{z_c}(y - p_y) \tag{1.15}$$

Due to their simplicity and linearity these equations have been the basis for a large amount of walking algorithms for the past several decades. One of the earliest and most successful algorithms was Kajita's preview controller in [43]. They used an LQR to continuously solve for bipedal trajectories and footstep locations over a finite time horizon. In [45] Kalakrishnan et al. was able to embed the ZMP dynamics into a quadratic program that allowed for solutions online of a quadruped going over uneven terrain.

If the Z height of the robot is not fixed to a plane then the z acceleration needs to be accounted for, the resultant dynamics are shown in (1.16) and (1.17). These equations allow for richer solutions but they are now no longer linear or convex.

$$\ddot{x} = \frac{g + \ddot{z}}{z}(x - p_x) \tag{1.16}$$

$$\ddot{y} = \frac{g + \ddot{z}}{z}(y - p_y) \tag{1.17}$$

Typically when ZMP dynamics are used in a TO problem a constraint must be added that explicitly constrains the ZMP point to be within the support polygon. This constraint boils down to determining whether a point is within a polygon. Although there are many algorithms for determining if a point is within a polygon most do not easily form an inequality constraint that is required for an optimization problem. For this reason one of the most common ways of encoding this constraint is by using the angle summation test. The first step of the test is to calculate the angle between the two vectors created by the vector going from the test point p_Z and one of the vertexes p_i and the vector going from vertex p_i to it's next adjacent vertex p_{i+1}. Repeat this process for all vertex going from one vertex to the next adjacent one. If the test point is within the support polygon than all resultant angles will have the same sign, fig. 1.3.

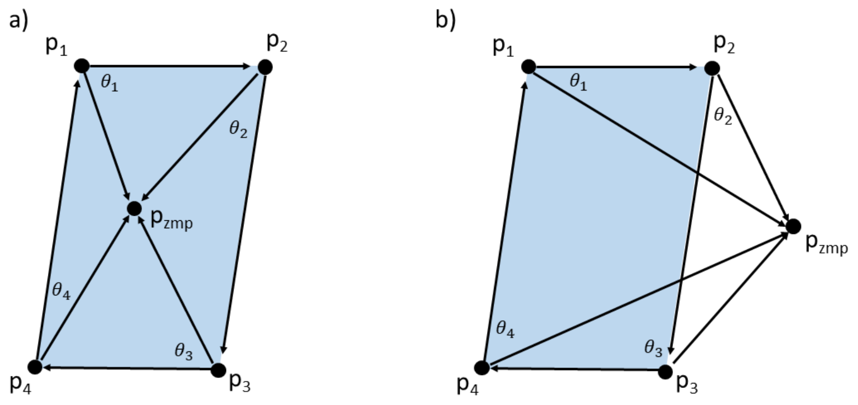


Figure 1.3: a) When the ZMP point is within the support polygon all angles have the same sign. b) When the ZMP point is outside of the support polygon the resultant angles will have different signs.

This formulation is relatively easy to encode however it has many shortcomings. In order to calculate the angle between the vectors trigonometric functions are required which make all the constraints non-linear. If the feet cross the order in which the vectors are created must be changed which adds a non-negligible amount of complexity to the formulation of the problem. And finally this method does not work if the support polygon becomes a line or a point which happens frequently if you assume the robot has point feet.

1.2.3 Special Orthogonal Group SO(3)

The following section provides a brief background on rigid body rotations. For a more through discussion on rigid body rations please see [56], [17], [72]. In robotic systems there are typically two coordinate frames that are of interest (and possibly more); the Inertial frame and the Body frame. The Inertial frame is a fixed frame typically setup with the Z axis inline with the gravity vector whereas the Body frame is attached to the robot. In this work all frames will be described using a right-handed convection. It is useful to be able to describe the relationship between these two frames because this will define how the robot is moving through space. Figure 1.4 illustrates a rigid body described in both frames. Letting

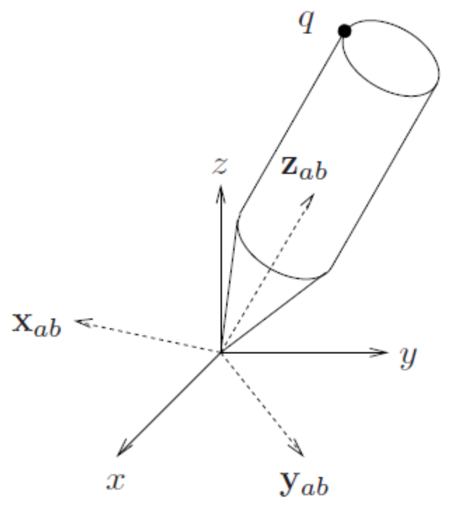


Figure 1.4: Rotation of a rigid object. The Inertial frame is represented by the solid coordinate frame while the Body frame is represented by the dotted coordinate frame. This image was taken from [56]

$\boldsymbol{x}_{ab}, \boldsymbol{y}_{ab}, \boldsymbol{z}_{ab}, \in \mathbb{R}^3$ be the principal axes of the Body frame relative to the Inertial frame, it follows that we can create a 3×3 matrix by stacking the vectors as such:

$$\boldsymbol{R}_{IB} = [\boldsymbol{x}_{ab} \quad \boldsymbol{y}_{ab} \quad \boldsymbol{z}_{ab}] \tag{1.18}$$

The formulation (1.18) is a *rotation matrix* which belongs to the *special orthogonal group* in $\mathbb{R}^3$ or $SO(3)$ for short. Rotation matrices are a special group that rotate a vector in $\mathbb{R}^3$ but preserve its magnitude. The following are mathematical properties associated with rotation matrices:

$$\mathbf{det}\boldsymbol{R} = 1 \tag{1.19}$$

$$\boldsymbol{R}\boldsymbol{R}^{\boldsymbol{T}} = \boldsymbol{R}^{\boldsymbol{T}}\boldsymbol{R} = \boldsymbol{I} \tag{1.20}$$

$$\boldsymbol{R_1}, \boldsymbol{R_2} \in SO(3), \mathbf{then} \quad \boldsymbol{R_1}\boldsymbol{R_2} \in SO(3) \tag{1.21}$$

From (1.20) we see that the inverse of a rotation matrix is just the transpose of the original rotation matrix, and from (1.21) we see that the product of two rotation matrices results in a rotation matrix. Using these properties we can easily describe a vector in $\mathbb{R}^3$ in any frame using a combination of rotation matrices. Consider a point $\boldsymbol{p}_a$ with the coordinates $\boldsymbol{p}_a = [x_a, y_z, z_a]^T$ described in Frame A. Equation (1.22) describes how to represent point $\boldsymbol{p}_a$ relative to Frame B using the rotation matrix $\boldsymbol{R}_{ba}$ which describes the rotation from the A frame to the B frame. Going further (1.23) describes how to represent point $\boldsymbol{p}_a$ relative to Frame C, notice that the combination of the rotation matrices describing the mapping from frame A to frame B and the mapping from frame B to frame C results in the rotation matrix that directly describes the mapping from Frame A to Frame C.

$$\boldsymbol{p}_b = \boldsymbol{R}_{ba}\boldsymbol{p}_a \tag{1.22}$$

$$\boldsymbol{p}_c = \boldsymbol{R}_{cb}\boldsymbol{p}_b = \boldsymbol{R}_{cb}\boldsymbol{R}_{ba}\boldsymbol{p}_a = \boldsymbol{R}_{ca}\boldsymbol{p}_a \tag{1.23}$$

Going back to frame A from frame C we can use (1.20) to describe the inverse of the mapping from Frame A to Frame C using the known rotation matrices.

$$\boldsymbol{p}_a = \boldsymbol{R}_{ca}^{-1}\boldsymbol{p}_c = \boldsymbol{R}_{ca}^{T}\boldsymbol{p}_c = \boldsymbol{R}_{ba}^{T}\boldsymbol{R}_{cb}^{T}\boldsymbol{p}_c \tag{1.24}$$

1.2.3.1 Euler angles

One way to represent the mapping from one frame to another is by perform a series of three rotations with each rotation only being performed about a single axis. Equations 1.25-1.26 describe the three possible pure rotations about the Z, Y, and X axes respectively. Where ψ represents the rotation angle about the Z axis, θ represents the rotation angle about the Y axis, and ϕ represents the rotation angle about the X axis. These three angles are refered to as the Euler angles as they are the three angles required to adequately describe a unique rotation matrix.

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.25}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{1.26}$$

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{1.27}$$

There are many different combinations of pure rotation matrices that can be used to describe the same rotation matrix. For example a Z-Y-Z described by (1.28) represents a rotation about the Z axis of frame B, then a rotation about the Y axis of the resultant frame, and finally another rotation about the Z axis of the resultant frame.

$$\boldsymbol{R_{wb}} = \boldsymbol{R_z}(\psi)\boldsymbol{R_y}(\theta)\boldsymbol{R_z}(\phi) \tag{1.28}$$

This combination of rotations will result in a unique set of Euler angles associated with the Z-Y-Z convention. It is important to note that a different set of Euler angles would be used with a different convention such as X-Y-Z. It is important to not mix conventions while performing calculations. For the remainder of this work all Euler angles will use the convention of Z-Y-X as described by (1.29).

$$\boldsymbol{R_{wb}} = \boldsymbol{R_z}(\psi)\boldsymbol{R_y}(\theta)\boldsymbol{R_x}(\phi) \tag{1.29}$$

1.2.3.2 Exponential coordinates

Exponential coordinates breakdown a given rotation into two parts, $\omega \in \mathbb{R}^3$ the unit vector that describes the axis by which the rotation occurs and $\theta \in \mathbb{R}$ the angle of rotation about this axis. The derivation for the exponential coordinate representation of a rotation matrix is provided in [17] and re-derived here for completeness.

Considering a point q attached to rigid body rotating at a constant unit angular rate of ω. Then the velocity of that point can be described as follows,

$$\dot{q}(t) = \omega \times q(t) = \hat{\omega} q(t) \tag{1.30}$$

where, the notation $\hat{\omega}$ represents the mapping from $\mathbb{R}^3$ to a skew symmetric matrix in $\mathbb{R}^{3\times 3}$ as shown in (1.31).

$$\begin{gathered} \boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T \\ \hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \end{gathered} \tag{1.31}$$

Equation (1.30) is a standard time-invariant linear differential equation which provides the following solution:

$$q(t) = e^{\hat{\omega} t} q(0) \tag{1.32}$$

If the rigid body is rotated about the axis ω at unit velocity for θ units the resultant rotation matrix can be represented by (1.33).

$$\boldsymbol{R}(\boldsymbol{\omega}, \theta) = e^{\hat{\boldsymbol{\omega}} \theta} \tag{1.33}$$

Expanding out the derived matrix exponential results in the following infinite series.

$$e^{\hat{\boldsymbol{\omega}} \theta} = \boldsymbol{I} + \theta \hat{\boldsymbol{\omega}} + \frac{\theta^2}{2!} \hat{\boldsymbol{\omega}}^2 + \frac{\theta^3}{3!} \hat{\boldsymbol{\omega}}^3 + ... \tag{1.34}$$

Unfortunately, this infinite series is not in a form that is useful for practical calculations. However, using a couple of linear algebra operations the original infinite series can be rearranged into a sum of two infinite alternating series.

$$e^{\hat{\boldsymbol{\omega}}\theta} = \boldsymbol{I} + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - ...\right)\hat{\boldsymbol{\omega}} + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + ...\right)\hat{\boldsymbol{\omega}}^2 \tag{1.35}$$

From this form we can finally write the matrix exponential into the *Rodrigues' formula*,

$$e^{\hat{\boldsymbol{\omega}}\theta} = \boldsymbol{I} + \hat{\boldsymbol{\omega}}\sin\theta + \hat{\boldsymbol{\omega}}^2(1-\cos\theta) \tag{1.36}$$

1.3 Organization and Contributions

This work focuses on expanding the capabilities of quadrupedal robots from the current state of the art through innovative mechanical design and real-time optimization techniques. Chapter 2 introduces the autonomous personal helper robot with enhanced dynamics (ALPHRED). The ALPHRED platform uses a unique kinematic design that places the limbs in a radially symmetric configuration rather than an elongated configuration parallel to the sagittal plane. In addition, the traditional quadrupedal robot has a hip roll actuator as the first degree of freedom whereas ALPHRED uses a hip yaw degree of freedom. Chapter 2 discusses how these kinematic changes allow ALPHRED to be a multi-modal robot, meaning that ALPHRED has many different configurations and modes of operations to handle a variety of different tasks.

Chapter 3 explains the high level trajectory planner developed for the ALPHRED platform. This chapter expands a 2D non-linear program that uses ZMP dynamics to plan for a number of footsteps into a 3D planner that is now capable of handling elevation changes.

This algorithm was further advanced by developing a method to incorporate footstep information from a vision algorithm in order to plan both footsteps and CoM trajectories. Traditional footstep planners use mixed-integer optimization to plan for discrete footstep plans, whereas the method developed in this chapter was able to formulate the problem into a continuous non-linear optimization problem.

Chapter 4 goes on to discus the low level control used to track the desired motion of the robot. Due to the discrete changes that occur as a result of legged locomotion each limb is controlled using a finite state machine. This methodology allowed for each leg to individually monitor when the leg should be in either stance or swing phases and what the actual state of the limb was. Utilizing this structure two different tracking controllers were developed. The first was a simple position controller that would command the feet to match the desired velocity of the CoM while adjusting the length of each limb to account for orientation errors. The second method expands on a torque controlled model predictive controller (MPC) that greatly simplifies the dynamics of the robot to plan for ground reaction forces over a finite time horizon. This work was able to reformulate the problem in order to solve the problem at +250Hz whereas the previous formulation solved at 50Hz. In addition, the dynamic model used was expanded to account for known swing leg dynamics which was ignored in the previous implementation.

Chapter 5 presents the manipulation techniques developed for the ALPHRED platform. Due to the fact that there is no other robotic platform that uses ALPHRED's unique kinematic design, all new techniques were required to allow ALPHRED to pick up packages of all sizes. First, custom end-effectors were developed to provide ALPHRED with an additional degree of freedom to complement the degrees of freedom that ALPHRED already posessed. Next, a pseudo-force controller was developed to ensure that the package was securely clamped during the entire transition from the ground to the top of the robotic platform. Finally, chapter 6 summarizes the work presented in this dissertation and discusses the implications it may have on future works.

CHAPTER 2

ALPHRED PLATFORM

The ALPHRED robotic platform started from ALPHRED-1, a position controlled quadrupedal robot. ALPHRED-1 has four radially symmetric 3 DoF limbs, with two dual MX-106 actuators at the hip and one dual MX-106 at the knee. A Typical quadrupedal robot has 3 DoF legs with the first joint controlling lateral ab/adduction (typically achieved with a hip roll), followed by the second joint controlling the hip pitch, and the third joint controlling the knee pitch. ALPHRED uses a unique kinematic design by switching the traditional lateral ab/adduction for the control of the medial rotation by using a hip yaw actuator. In addition, traditional quadrupeds typically have a rectangular limb configuration giving them a natural front and back along the sagital plane. ALPHRED uses a radially symmetric configuration removing the reliance on a front and back. These simple changes give ALPHRED a unique and greater kinematic workspace compared to traditional quadrupeds. For example, each of ALPHRED's limbs are capable of reaching from the belly of the robot to the top of the body in a single continuous motion at any hip yaw angle. Additionally, because of ALPHRED's radial symmetry, any two opposite limbs have the range of motion to become parallel so that they can then perform dual limb manipulation tasks. To the authors knowledge there is no other quadrupedal robotic platform with this type of kinematic range. ALPHRED-1

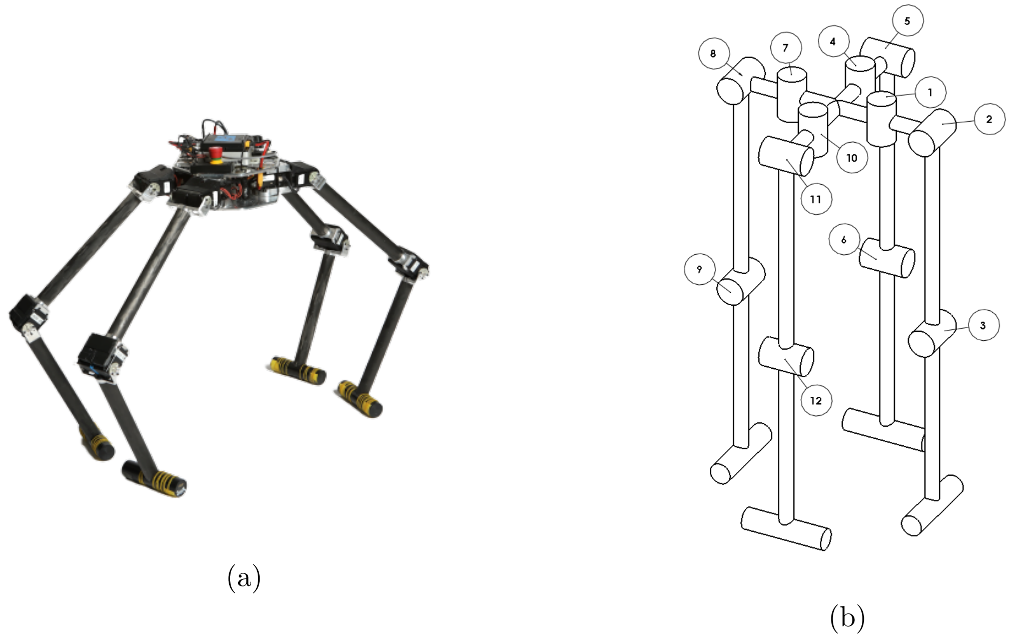


Figure 2.1: a) Image of the full ALPHRED-1 platform. b) Bubble diagram of ALPHRED-1 joint layout, see Table 2.1 for range of motion

is equipped with bar feet which increase the size of the support polygon providing the capability for the robot to balance on two opposite limbs allowing for the remaining two limbs to be used for simple manipulation tasks.

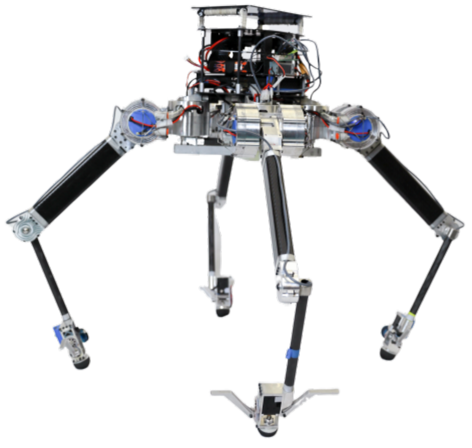
Although, these kinematic changes improve the robot's workspace they have negative impacts on the robot's dynamics. When a traditional quadrupedal robot is nominally traversing forward the lateral ab/adduction actuation is not used and only the hip pitch and knee pitch is required. The lateral ab/adduction is only used for disturbance mitigation and turning. Legged robotic limbs are built to reduce inertia of the limbs by moving all of the actuators to the hip and using a transmission system to control the hip knee joint. With this design and nominal forward locomotion only the links of the leg will be moving resulting in small

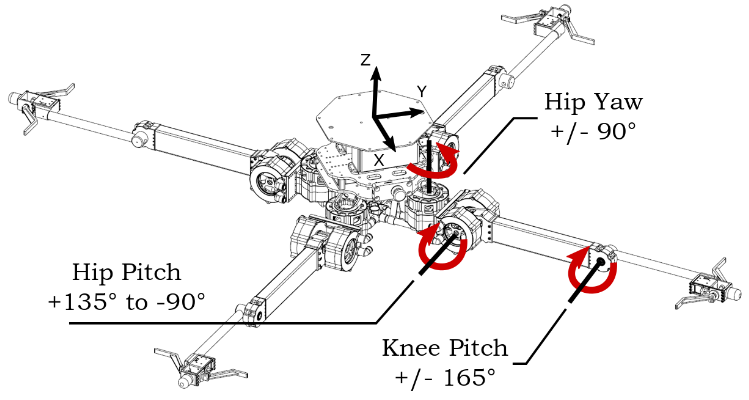
Table 2.1: Range of motion for each joint.

Joints	**min**	**max**
hip yaw (1,4,7,10)	-92°	92°
hip pitch (2,5,8,11)	-92°	92°
knee pitch (3,6,9,12)	-92°	92°

forces and moments acting on the body due to leg swings. However, this is not the case for ALPHRED because the limbs are no longer aligned with the direction of motion, for this reason the hip yaw actuator must be used through out the swing leg motion causing a much larger inertial impact compared to that of a traditional quadruped. In addition, it is well understood that for high speed gaits such as galloping the feet should align along the direction of travel. The main reason for this is that at high speeds it is desirable for all forces to be used to keep the robot upright and to propel the robot forward. When the feet do not align with the plane of the CoM then some of the forces are used to push the robot in the direction perpendicular to the direction the robot is traveling, which can cause a swaying motion and reducing the efficiency of the gait. ALPHRED is not capable of placing the feet along the line of travel due to collisions between the two adjacent hip actuators. Although, ALPHRED's unique kinematic design results in decreased dynamic performance ALPHRED was still able to achieve a trotting gait of 1.7m/s and perform other forms of locomotion and manipulation that other quadrupeds are not capable of.

Following the success of ALPHRED-1, ALPHRED-2 was developed as a high performance version (Fig. 2.2). ALPHRED-2 used the same novel kinematic design of the radially


(a)



(b)

Figure 2.2: a) Image of the full ALPHRED-2 platform. b) Kinematic range of motion of the ALPHRED-2 platform.

symmetric limb layout combined with the non-traditional hip yaw actuator rather than a hip roll actuator. However, the dual MX-106's were switched out for Backdrivable Electromagnetic Actuator for Robotics (BEAR) [89]. The BEAR actuators are high performance proprioceptive actuators described in more detail in Section 2.2. ALPHRED-1's limbs were constructed in a serial fashion with all actuators being colocated with the joint that was being controlled. To reduce the inertia of the limb ALPHRED-2 uses a parallel configuration for the hip and knee joints with the knee joint being controlled in a non-colocated fashion via a belt drive (Fig. 2.3). The torque at the knee is created by the difference in torque between the knee actuator and the hip actuator. The parallel configuration was chosen to further reduce the inertia of the leg; In a serial configuration the entire knee actuator would rotate with the leg, whereas in the parallel configuration both the hip pitch and knee pitch actuators are stationary during extension and retraction of the limb. The belt drive helped to further increase the range of motion of the ALPHRED platform, whereas with ALPHRED-1 the knee joint's range of motion is ±90° the belt allows for almost a full 360° of rotation (Fig. 2.2b). Additionally, the bar feet were replaced with point feet to allow for locomotion over uneven terrain. However, bi-pedal operation is still achievable due to an additional degree of freedom provided at the feet. Attached to the additional degree of freedom are structural wings that can swing down to increase the support polygon during bi-pedal operation. Different end-effectors can also be attached to the feet to improve manipulation capabilities, this is discussed further in Chapter 5.

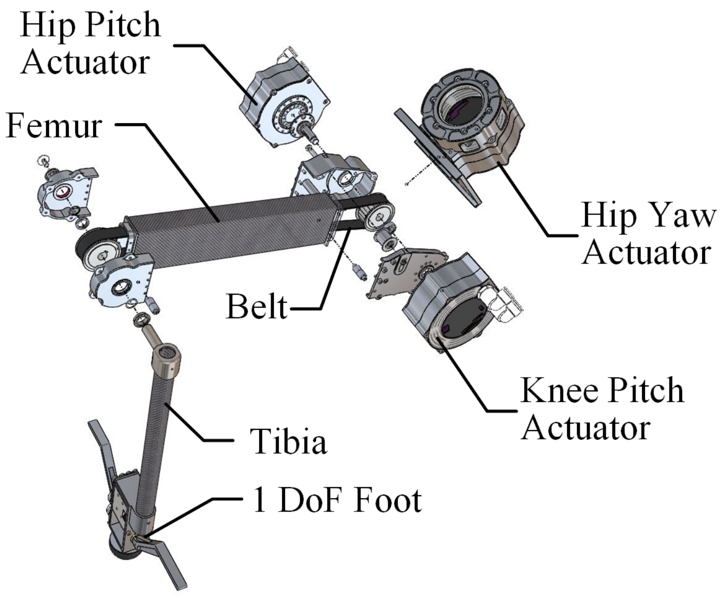


Figure 2.3: Exploded view of a single ALPHRED-2 limb.

2.1 Modes of Operation

One of the most unique features of the ALPHRED platform is the robot's ability to transform into different modes in order to handle a multitude of different tasks. This is only achievable due to ALPHRED's unique kinematic design.

Omni-directional Mode is when the limbs are in a broad radially symmetric configuration, Fig. 2.4a. In this mode the robot has a large support polygon providing stability in all directions. This mode is the nominal configuration that is used to transition between all other modes. The broad stance provides for slow and stable locomotion over rough terrain. In addition, this mode allows for small adjustments in the robot's position or pose to help re-orient the robot for manipulation tasks.

Dynamic Mode is when the limbs are configured in a traditional quadrupedal stance symmetrical about the sagittal plane and the coronal plane, with the stance elongated parallel to the sagittal plane (Fig. 2.4c). This mode is used for fast and dynamic locomotion similar to traditional quadrupeds. In this mode ALPHRED is capable of a 1.7m/s trot, as well as an amble gait and pace gait.

Caster Mode is when the robot lowers itself onto four caster wheels attached to it's body (Fig. 2.4d). The robot is then able to use its feet to push itself along similar to how a skate boarder propels themselves forward. Because of ALPHRED's kinematic design the feet are always able to stay perpendicular to the ground allowing for the robot to maintain a sufficient normal force without slipping. This mode is extremely efficient and is used when ALPHRED must travel long distances across flat terrian. The robot is able to travel at 2.1m/s with a Cost of Transport (CoT) of 0.55 (the CoT metric used is the same one defined in [10]), compared to that of ALPHRED's dynamic trot of 1.7m/s with a CoT of 2.6.

Tripod Mode is when ALPHRED positions three of it's limbs into an equilateral triangle formation while lifting the remaining limb off of the ground (Fig. 2.4b). The three supporting limbs create a strong base with a large support polygon while the limb in the air can be used for manipulation tasks that require large forces. To demonstrate the stability of tripod mode ALPHRED has punched through a wooden board used in karate classes.

Bipedal Mode is when the structural wings on two opposing limbs are commanded into the down position, creating a large enough support polygon for the robot to balance on the two limbs. The remaining two limbs are then lifted off the ground and can be used for dual manipulation tasks (Fig. 2.4f). In this mode ALPHRED can pick up boxes of various size and weights from heights up to 1 meter off of the ground.

2.2 BEAR Actuators

The BEAR modules used on the ALPHRED-2 platform are proprioceptive actuators that combine high torque brushless DC (BLDC) motors with a single phase planetary gearbox, Fig. 2.5). The design philosophy of the BEAR modules is similar to the design concepts found in [50], [71], [85]. The BEAR modules were designed to maximize the torque density of the actuator (the ratio of the peak torque to the mass of the actuator) while still preserving the transmission transparency. For this reason, a large air gap radius BLDC was chosen for the motor of the actuator rather than a small radius motor as found in standard high gear reduction servo motors. Additionally, only a single stage planetary gear box was used with a 10:1 reduction ratio to allow for proprioceptive force sensing by reducing the reflected inertia and the friction of the system.

The benefits of preserving transmission transparency is most readily apparent from the

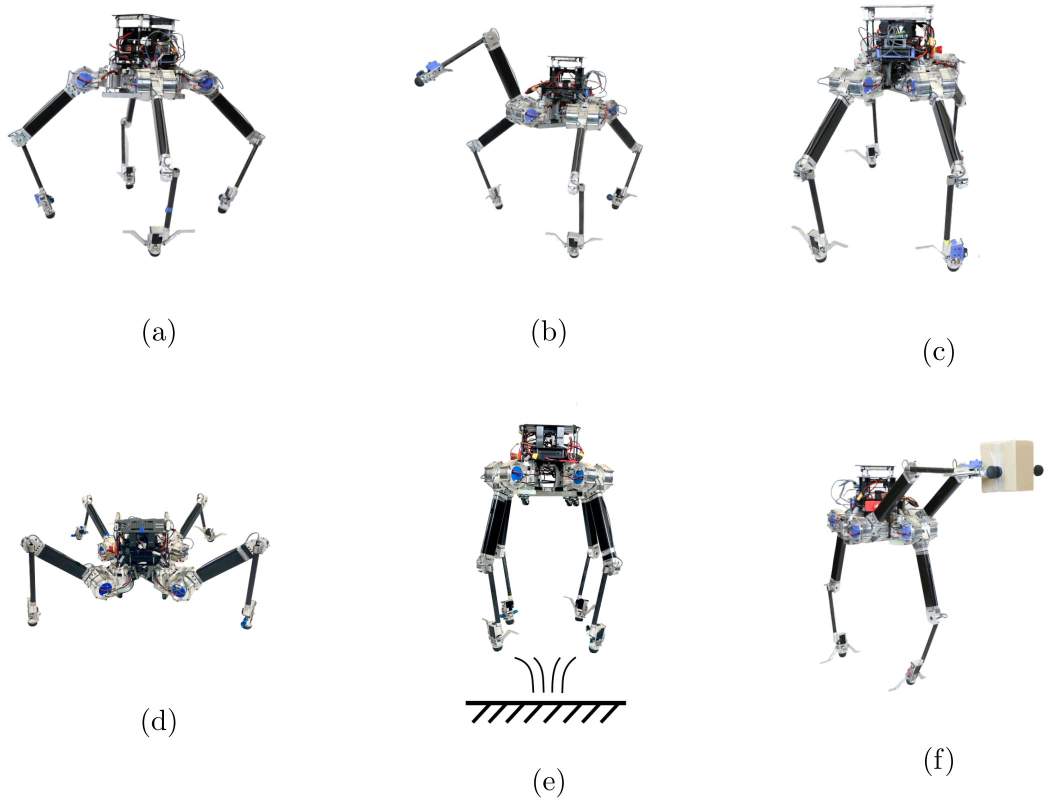


Figure 2.4: a) Symmetric stable mode, b) Tripod single manipulation mode, c) Dynamic mode, d) Caster/wheeled mode, e) Jumping/pronking mode, f) Biped dual manipulation mode

Fig. 2.6 which summarizes a stall torque experiment between a MX-106 actuator with a gear reduction of 225:1 and the BEAR module. During this experiment each actuator was driven with a constant current and the resultant stall torque was measured and is shown with the blue triangles as the ”Driven” torque curve. Next an external torque is applied to the actuator until the position changes and the torque is recorded and is shown with the orange dots as the ”Back-driven” torque curve. As can be seen from the comparison between Fig. 2.6a and Fig. 2.6b, the BEAR module maintains a constant torque whereas the MX-106

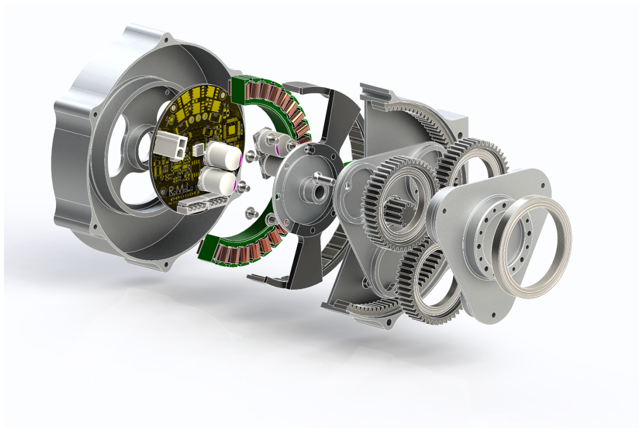
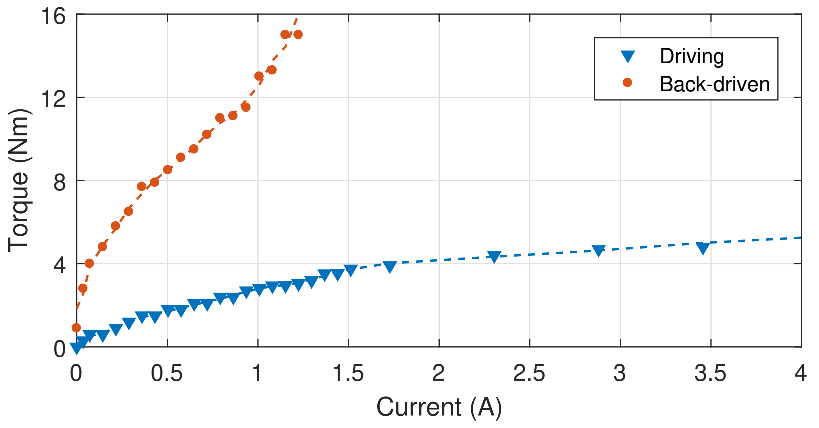


Figure 2.5: BEAR module exploded view of mechanical design.

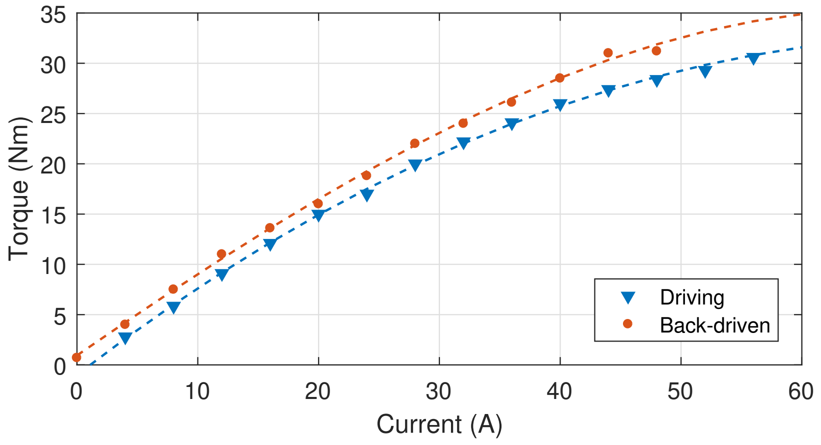
has vastly different torque curves depending on whether it is being back-driven or not. A curve can be fit to the data from Fig. 2.6b which can be used to approximate the torque by measuring the current in the windings.

One of the main drawbacks of these types of actuators is the excessive Joule heating that is produced during operation [38]. If the excess heat is not managed properly the actuator can overheat causing damage to the windings and electronics. The continuous torque of the actuator is determined by how much current can be applied continuously without causing the motor to overheat. In order to increase the continuous torque of the BEAR modules the motor housing was built as a heat sink with the option of using liquid cooling to improve the heat transfer properties of the the module. As shown in Fig. 2.7 the stator of the motor is connected to the motor housing via thermal paste analogous to the pastes used for CPU's. The opposite side of the housing connected to the stator is designed with fins to increase the surface area to increase heat transfer either to air or water if used. Table 2.2 summarizes the specifications including the improved continuous torque if liquid cooling is used.

[1]Varies depending on the cooling system



(a) Torque curve for high reduction MX-106 actuator



(b) Torque curve for BEAR

Figure 2.6: Comparing stall torque curves for high gear reduction gearboxes vs low gear reduction gearboxes.

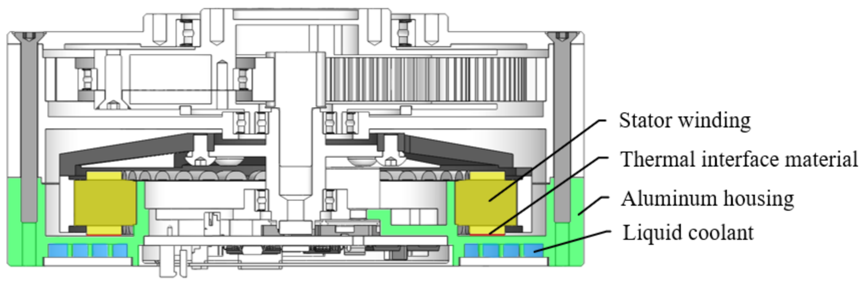


Figure 2.7: Cross sectional view of the BEAR module showing the liquid cooling section.

Table 2.2: BEAR Specs

Mass (g)	670
Gear Reduction	10:1
Voltage (V)	30
Max Current (A)	60
Peak Torque (Nm)	32
Cont. Torque (air cooled, Nm)	7.8
Cont. Torque (liquid cooled, Nm)[1]	~ 21
Max Velocity (rpm)	300

2.3 System Architecture

ALPHRED-2 uses an off-the-shelf Intel NUC equipped with an Intel Core i7-6670HQ @ 2.60 GHz with 32 GB of RAM. The control architecture is broken up into five separate processes, as shown in Fig. 2.8, that run in parallel in order to allow for tele-operation of the robot. All communication between threads is done with a custom shared memory library that allows for sharing of floating point arrays. The user input process runs a C++ TCP server that communicates with a GDP handheld computer at 40 Hz. The raw inputs from the GPD are processed and then passed as a desired velocity vector $(\dot{x}, \dot{y})$ and angular yaw rate $(\dot{\psi})$ to the FSM/Trajectory Planning thread. The FSM/Trajectory Planning thread runs a Finite State Machine (FSM) that is a high level controller that transitions the robot from task to task. Once the robot has transitioned into a locomotion task the FSM transitions to the Trajectory Planning state where the desired user inputs are used to create a reference trajectory. The trajectory planner either propagates the desired velocities into a straight line reference trajectory at 700Hz or uses a non-linear TO algorithm to solve for dynamically feasible trajectories, both methods are written in Python 2.7. The operator is able to switch between the two trajectory planning methods. The non-linear TO algorithms are discussed in detail in Chapter 3. Next, the reference trajectory is then passed to the motion controller thread which tries to track the given reference trajectory. This is done either via a 700 Hz position PD controller that controls limb length to try to mitigate disturbances or from a QP model predictive controller (MPC) that controls the ground reaction forces of the limbs in contact and runs at 150 Hz. Both of these methods are written in C++ and are described in greater detail in Chapter 4. The output of the motion controller is then passed to the motor controller thread. This is a Python thread that runs at 1000Hz, which handles all the communication between the BEAR modules and the on board computer. A state estimator thread is continuously ran at 400Hz providing full state feedback to the Trajecory Planning and Motion Controller threads. The following sub-section provides a brief summary of the state estimator that is used on the ALPHRED-2 platform.

2.3.1 State Estimation

The state estimation algorithms used on ALPHRED-2 are an Extended Kalman Filter (EKF) derived in [15] and and Unscented Kalman Filter (UKF) derived in [14]. Both filters blend the measurements taken from the IMU and the joint encoders of the limbs. IMU's provide linear acceleration and angular rate measurements. By integrating the linear acceleration and angular rate the velocity, position, and orientation of the robot can be estimated. By exploiting the unique way in which legged robots traverse through their environment another

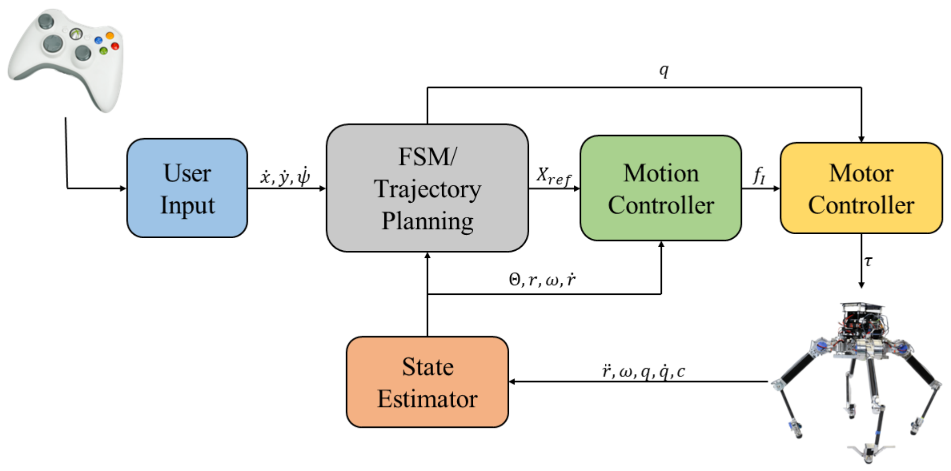


Figure 2.8: Control system block diagram. Each block runs a separate process with all inter-process communication being handled by a custom shared memory library.

measurement of the robot's position and velocity can be recovered using the kinematics of the robot. Assuming that a foot does not slip we can back out what the position and velocity of the robot by using the forward kinematics and the joint encoders for each limb in contact with the ground. Noise in the sensors will cause small errors in the measurements even with a high accuracy IMU's and joint encoders. Integrating these small errors will result in large errors of the estimated state, which is why a Kalmand filter is used to stabilize the state estimation. Equations (2.1)-(2.6) show the prediction step of both Kalman filters.

$$\hat{r}_{k+1} = \hat{r}_k + \Delta t \hat{v}_k + \frac{\Delta t^2}{2}[\hat{R}_k^T(a_k - \hat{b}_{a,k}) + g] \tag{2.1}$$

$$\hat{v}_{k+1} = \hat{v}_k + \Delta t[\hat{R}_k^T(a_k - \hat{b}_{a,k}) + g] \tag{2.2}$$

$$\hat{q}_{k+1} = exp[\Delta t(\omega_k - \hat{b}_{\omega,k})] \otimes \hat{q}_k \tag{2.3}$$

$$\hat{p}_{i,k+1} = \hat{p}_{i,k+1} \quad \forall i \in \{1, ..., N\} \tag{2.4}$$

$$\hat{b}_{a,k+1} = \hat{b}_{a,k} \tag{2.5}$$

$$\hat{b}_{\omega,k+1} = \hat{b}_{\omega,k} \tag{2.6}$$

Where ˆ means the estimated state, r is the linear position, v is the linear velocity, a is the linear acceleration, q is the quaternion, ω is the angular rate, p_i is the position of foot i, b_a is the bias in the angular acceleration measurements, and b_ω is the bias in the angular rate measurements. The rotation from the inertial frame to the body frame is represented by the 3×3 rotation matrix R. To integrate the quaternion the operators $exp()$ and $\otimes$ are used which are defined in [77]. The measurement is the error between the measured distance to the foot of the robot from the joint encoders and the forward kinematics and the distance to the foot measured by the prediction state, shown in (2.7).

$$y_k = \begin{bmatrix} s_{1,k} - \hat{R}_k(\hat{p}_{1,k} - \hat{r}_k) \\ \vdots \\ s_{N,k} - \hat{R}_k(\hat{p}_{N,k} - \hat{r}_k) \end{bmatrix} \tag{2.7}$$

Where s_i represents the measurement from the forward kinematics of foot i. At this point the update step is performed which updates the estimation of the states and propagates the co-variance matrices. For the EKF this step is done using a local linear approximation and for the UKF a non-linear propagation is done multiple times and then sampled to more accurately approximate the non-linear effects.

CHAPTER 3

TRAJECTORY PLANNING

The following chapter discusses the algorithms that were developed and implemented on the ALPHRED V2 platform. The developed algorithms produce CM trajectories and footsteps that are implemented on the robot in an open-loop fashion. The ALPHRED V2 platform [36] is a position controlled robot without force feedback of any kind which only allows for open-loop control of the overall trajectory of the robot. Despite this limitation, these algorithms were still able to create a trotting gait of 1.5 m/s and motion to step up and over an 8 inch tall obstacle.

The following TO algorithms use a direct collocation based formulation to solve for trajectories online with time horizons beyond that of one gait cycle. All methods were ran on the ALPHRED V2 platform. The first method in Section 3.2 outlines a convex problem that can solve for CM trajectories given a footstep plan. This method uses the vertex based ZMP constraint first derived in [86] to allow for quick solution of trajectories even when the support polygon become a line or point. This method expands on the previous work by using the full 3D ZMP dynamics and adding a cost to the overall acceleration of the trajectory. These additions expanded the algorithms capabilities to uneven terrain and improved performance with gaits that have adjacent support polygons that do not intersect, like bounding and pacing.

Method two, discussed in Section 3.4, builds off of the first method by replacing the convex problem with a NLP that solves for both footsteps and CM trajectories using both the dynamics of the robot and vision feedback to decide where to place the feet. A stitching method was created in order to allow for tele-operation of the robot. Most trajectory optimization algorithms separate the planning of the CoM trajectory and the planning of the the footsteps into two different optimization problems. A footstep plan is a descrete discontinous list of positions that is tricky to incorporate into the overall optimization problem. In [18] Deits et al. developed a mixed-integer quadratically-constrained quadratic program (MIQCQP) to solve for an optimal footstep plan. In [54] Kuindersman et al. developed a stabilizing QP controller for dynamic walking that would use the MIQCQP as the input to the stabilizing controller. These two algorithms were used by the MIT VRC team at the Darpa Robotics Challenge. Similarly, in [46], Schaal used a machine learning to learn a footstep planner for extremely uneven terrain for a quadrupedal robot. The output of this footstep planner was passed to a cascade of other planners and controllers that would control the pose and CoM of the robot. Winkler expanded upon these works by combining the optimization of the footsteps and the CoM trajectory into one NLP in [86] and [88]. In both of these works Winkler showed that the footstep locations can be used in to enhance the dynamics of the robot. However, both Winkler's algorithms assumed that a complete terrain map was known a-priori and all terrain was exactly the same. To the best of the author's knowledge this is the first algorithm that optimizes for both footstep locations and CoM trajectory while being able to convey important terrain information to the algorithm in real time.

3.1 Footstep Planning

For legged system (biological or robotic) the footstep timing sequence or gait, plays a large role in how effectively a system is able to traverse various terrains. For quadrupedal systems there are many different types of gaits with common ones being: A walk with 3 feet are on the ground at a time, a pace with 2 feet are on the ground at a time switching across the sagittal plane, a trot with 2 feet are on the ground switching along the diagonal, and a gallop with one foot is on the ground at a time. Different gaits minimize energy consumption at different velocities [37], making it ideal for a robotic system to be able to easily transition between different gaits.

To define a gait for a given system the number of contacts at a given time must be defined. For a multi-legged system the number of possible contacts becomes increasingly large and difficult to plan. However, if we only look at a single limb only two elements are required to define its contact schedule; T_{stance} the time that the leg will be in stance phase and T_{swing} the time the leg will be in swing phase. By planning the contact schedule for each foot individually the developer can quickly and easily come up with any gait desired [88].

The footstep planner will return a list of times and positions for each leg to be passed to the optimization algorithm. Each plan starts at time zero and goes until time t_{2n-1}.

$$\begin{aligned} \boldsymbol{T}_i &= [t_1, t_2, ..., t_{2n-1}] \\ \boldsymbol{P}_i &= [p_1, ..., p_n] \end{aligned} \tag{3.1}$$

Where $i \in \{1, 2, 3, 4\}$ is the i-th leg, $t_j \in \mathbb{R}$ is a time in seconds, and $f_j \in \mathbb{R}^3$ is a position vector. The entry f_1 for each leg is the current position of the foot. Whereas, t_1 is the time when the foot lifts of the ground and starts its swing phase, t_2 is when the foot touches down on the ground and starts its support phase. This alternating pattern continues until time t_{2n-1} where all odd valued times are liftoffs and even valued times are touchdowns. The position f_k corresponds to the position of the foot during the support time between $t_{2(k-1)}$ and t_{2k-1}. If foot i is initially in the swing phase than t_1 will be zero. This formulation makes it quite simple to change the gait of the robot by simply changing the time vectors for each leg. An indicator function can be derived from the time vector that indicates whether foot i is on the ground or not.

$$C_i(t) = \begin{cases} 1 & t_{2(k-1)} < t < t_{2k-1} \\ \\ 0 & otherwise \end{cases} \tag{3.2}$$

Where $C_i(t) = 0$ when the i-th foot is off of the ground.

The position of each foot is determined by combining Raibert [67] and Capture Point (CP) [65] heuristics similar to the technique used in [12]. All calculations for foot placement are done in the body frame.

$$\mathbf{p_j} = \underbrace{\mathbf{R}(\dot{\psi}_d \frac{1}{2} t_{2(j-1)})\mathbf{p_0} + \frac{1}{2} t_{2(j-1)} \upsilon_d}_{\text{Raibert}} + \underbrace{\frac{1}{2}\sqrt{\frac{p_z}{g}}(\upsilon - \upsilon_d)}_{\text{Capture Point}} \tag{3.3}$$

where $\mathbf{p} \in \mathbb{R}^2$ is the desired X and Y position for the next footstep location, p_z is the nominal height of the robot, g is the scalar value for gravity, $\upsilon \in \mathbb{R}^2$ is the current velocity of the body, $\upsilon_\mathbf{d} \in \mathbb{R}^2$ is the desired velocity in the body frame given by the operator, $\dot{\psi}_d$ is the desired yaw rate given by the operator, $\mathbf{p_0} \in \mathbb{R}^2$ is nominal position of the limb, and $\mathbf{R} \in \mathbb{R}^{3\times3}$ is a rotation matrix. In practice, the CP contribution is only used when $|\upsilon - \upsilon_d| > 0.15$ which helps correct major disturbances but does not come into play when operating normally. Additionally, CP only contributes to the first step in $\boldsymbol{P}$ because it is assumed that the controller will be able to follow the desired trajectory perfectly after the first step.

3.2 Center of Mass Trajectory Planning through Convex Optimization using Vertex Based ZMP Constraints

3.2.1 Problem Formulation

The problem is discretized into a discrete number of optimization parameters that fully describe the continuous time trajectory. The set w_c describes the x and y CM trajectory and w_u describes the loading condition of each foot.

$$\begin{aligned} w_c &= [a_{1,0}, ..., a_{1,4}, ..., a_{n,0}, ..., a_{n,4}]^T \\ w_u &= [\lambda_{1,1}, ..., \lambda_{1,4}, ..., \lambda_{n,0}, ..., \lambda_{n,4}]^T \end{aligned} \tag{3.4}$$

Where n is the number of polynomials that describe the trajectory, $a_{i,j} \in \mathbb{R}^2$ describes the i-th polynomial's j-th coefficient for both x and y, $\lambda_{i,j} \in \mathbb{R}$ describes the j-th foot's percentage of the total load at the time of the i-th polynomial. Using this parameterization of the problem it becomes relatively simple to compute all required dynamics of the robot at any given time. Given,

$$\boldsymbol{A}_i = [a_{i,0}, a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4}] \quad \boldsymbol{T} = [t^0, t^1, t^2, t^3, t^4]^T \tag{3.5}$$

the CM position, velocity, and acceleration can be computed by the following:

$$\begin{aligned}\boldsymbol{r}(t) &= \boldsymbol{A}_i\boldsymbol{T}\\ \dot{\boldsymbol{r}}(t) &= \boldsymbol{A}_i\dot{\boldsymbol{T}}\\ \ddot{\boldsymbol{r}}(t) &= \boldsymbol{A}_i\ddot{\boldsymbol{T}}\end{aligned} \tag{3.6}$$

The center of pressure (CoP) can also be calculated with the following formulation.

$$\boldsymbol{u}(t) = \sum_{f=1}^{4} \lambda_{i,f}\boldsymbol{p}_{m,f} \tag{3.7}$$

For example, if $\lambda_{1,1} = 1$ then at time t_1 foot 1 would support one hundred percent of the robot's weight which would result in the CoP being directly above foot 1. In Section 3.2.2 constraints are discussed that ensure that for any give time all of the λ's sum to 1.

Where, $\boldsymbol{p}_{m,f} \in \mathbb{R}^3$ is the 3D position vector that describes foot f's position at a given time.

3.2.2 Trajectory Optimization Problem

Figure 3.1 shows the overall structure of the convex program to produce CM trajectories for a given footstep plan. The TO problem will solve for coefficients of the CM trajectory polynomials (w_c) and foot loading conditions (w_u) that will minimize a cost on CM acceleration and conservative ZMP locations, while adhering multiple constraints to ensure feasibility.

The subsequent sections give extensive details to the costs and constraints of the TO program.

$$
\begin{aligned}
\min_{w_c, w_u} \quad & \|\ddot{r}(t)\|^2 && \text{(CM acceleration)} \\
& \|\lambda(t) - \lambda^*(t)\|^2 && \text{(Optimal ZMP location)} \\
\text{s.t.} \quad & && \\
& r(0) = r_0 && \text{(initial state)} \\
& r(T) = r_f && \text{(final state)} \\
& r(t_k^+) = r(t_k^-) && \text{(Continuity)} \\
& \ddot{r}(t) = f(r(t), u(t), p(t)) && \text{(ZMP dynamics)} \\
& |\boldsymbol{v}_i - r(t) + p_i(t)| > 0 && \text{(Kinematic model)} \\
& \textstyle\sum \lambda_i(t) = 1 && \text{(Total Loading)} \\
& 0 < \lambda_i(t) < c_i && \text{(Single Leg Load)}
\end{aligned}
$$

Figure 3.1: An overview of the trajectory optimization algorithm.

3.2.2.1 CM Acceleration Cost

The CM acceleration cost minimizes the total acceleration of the CM. Prioritizing this cost creates smooth natural motion while minimizing the effort required by the actuators.

$$\theta_a \int \|\ddot{\boldsymbol{r}}(t)\|^2 dt \tag{3.8}$$

In practice (3.9) was used to compute the CM acceleration cost by discritizing the continuous time integral along the CM trajectory.

$$\theta_a \sum_{t=0}^{2N+1} \|\ddot{\boldsymbol{r}}(t)\|^2 \tag{3.9}$$

Where, N is the number of polynomials and $\ddot{\boldsymbol{r}}(t)$ was calculated using (3.6). The cost was calculated at the beginning and middle of each polynomial. Note that because of the continuity constraints the beginning of polynomial $k+1$ is the same as the end of polynomial

k, thus the beginning, middle, and end of each polynomial was in essence used to determine the acceleration cost. The gain θ_a is used to change the priority level of this cost compared to the other costs of the TO problem. Currently θ_a is hand tuned.

3.2.2.2 Loading Cost

The optimal loading cost chooses conservative trajectories for the robot and was first introduced in [86]. This cost compares each foot's loading condition ($\lambda_{i,j}$) to the foot's optimal loading condition ($\lambda^*(i,t)$). The optimal loading condition balances the load between all feet that are currently on the ground. This will create a CoP position directly in the middle of the support polygon which can be thought of as the most stable position.

$$\begin{gathered}\theta_l \int \sum_{j=1}^{4} (\lambda_{i,j} - \lambda^*(j,t))^2 dt \\ \text{where,} \quad \lambda^*(j,t) = \frac{C_j(t)}{\sum_{i=1}^{4} C_i(t)}\end{gathered} \tag{3.10}$$

The implementation of (3.10) is done by discritizing the continuous time integral as shown in (3.11).

$$\theta_l \sum_{i=0}^{N} \sum_{j=1}^{4} [\lambda_{i,j} - \lambda^*(j,t(i))]^2 \tag{3.11}$$

Where, N is the total number of loading conditions the feet are broken up into over then entire trajectory. The gain θ_l is used to change the priority level of this cost compared to the other costs of the TO problem. Currently θ_l is hand tuned.

3.2.2.3 End Condition Constraints

The optimized values at time zero must be equal to the initial state r_0 and the optimized values at time T must be equal to the desired final state r_f

$$\begin{aligned} r(0) &= r_0 \\ \dot{r}(0) &= \dot{r}_0 \\ r(T) &= r_f \\ \dot{r}(T) &= \dot{r}_f \end{aligned} \tag{3.12}$$

3.2.2.4 Continuity

The following is an equality constraint that ensures continuity between adjacent polynomials.

$$\begin{aligned} \boldsymbol{A}_i \boldsymbol{T}_f - a_{i+1,0} &= 0 \\ \boldsymbol{A}_i \dot{\boldsymbol{T}}_f - a_{i+1,1} &= 0 \end{aligned} \tag{3.13}$$

Where, T_f corresponds to the time vector described in (3.5) when using polynomial i's final time t_f.

3.2.2.5 ZMP dynamics

The dynamic model used in this algorithm is the well known ZMP equation shown in (3.14). The values $\ddot{c}_z$ and c_z represent the Z acceleration and position of the robot respectively. These values are pre-computed based off of known elevation changes. The method for computing the z trajectories is discussed Section 3.2.4.

$$\ddot{\boldsymbol{r}}(t) = f(w_c, w_u, t) = \frac{g + \ddot{c}_z}{c_z}(\boldsymbol{r}(t) - \boldsymbol{u}(t)) \tag{3.14}$$

The equality constraint (3.15) ensures that the optimized trajectory satisfies the ZMP dynamics along the entire trajectory.

$$\int \boldsymbol{A}\ddot{\boldsymbol{T}} - f(w_c, w_u, t)dt = 0 \tag{3.15}$$

In order to implement the continuous integral in (3.15) the polynomial is checked at the two end points and the middle as shown in (3.16).

$$\boldsymbol{A}\ddot{\boldsymbol{T}} - f(w_c, w_u, t) = 0 \quad \forall t \in t_k, \frac{t_{k+1} - t_k}{2}, t_{k+1} \tag{3.16}$$

Simpson's rule states that if two fourth order polynomials are equal at the two end points and the middle than the error is $\propto (t_{k+1} - t_k)^5$ over the interval t_{k+1} to t_k. As long as the interval t_k to t_{k+1} is sufficiently small than the error will be small and the trajectory will be close enough to the desired ZMP dynamics. In order to increase the accuracy of the dynamics smaller intervals can be used, however the ZMP dynamics is a large simplification of the overall dynamics of the robot which will incur it's own errors. For this reason intervals of 20ms - 10ms were found to be sufficiently small.

3.2.2.6 Kinematic

The inequality constraint described by (3.17) ensures that the relationship between the CM position and the i-th foot position is bound by a box with dimensions r_{xy}. The value r_{xy} must be set based off of prior knowledge of the robotics kinematics.

$$-r_{xy} < \boldsymbol{v}_i - r(t) + p_i(t) < r_{xy} \tag{3.17}$$

Where $\boldsymbol{v}_i$ is the vector from the robot's CM to the i-th foot in the robot's nominal pose. Equation (3.17) never explicitly computes the forward kinematics or checks the joint limits. Instead, equation (3.17) decreases the complexity by implicitly ensuring joint limits by comparing the nominal length of the i-th limb to its current pose. Please note that this method does not guarantee that a joint limit is not violated but is still effective if r_{xy} is set conservatively.

3.2.2.7 ZMP constraint

This section describes the two constraints required to ensure that the ZMP is within the support polygon along the entire optimized trajectory. Winkler first derived this method, which he called the Vertex Based ZMP method, in [86]. This novel method improved upon standard methods in two ways: It is much simpler to implement and it is able to handle situations when the support polygon becomes a line or a point.

The inequality constraint described in (3.18) bounds all feet on the ground between 0 and 1 and all feet off of the ground are forced to be zero. This ensure that no leg in a swing phase carries any load. The equality constraint (3.19) makes certain that all of the leg's loading conditions add up to 1, meaning that all load bearing legs are supporting 100 percent of the forces on the CM.

$$0 < \lambda_{i,j} < C_i(t) \tag{3.18}$$

$$\sum \lambda_{i,j} = 1 \tag{3.19}$$

If both (3.18) and (3.19) are satisfied than the ZMP is defined to lie within the support polygon. Notice that this formulation does not explicitly check to see if the ZMP is within the support polygon like the Angle Summation Method but it implicitly ensures that this is always the case for any feasible set of decision variables.

3.2.3 Swing Leg Trajectory

Cycloidal functions were used to create the trajectories to move the foot from one position to the other. Two common approaches for swing leg trajectories are linear triangular profiles and Bezier curves: the first is discontinuous and creates jerky unstable motions whereas the second requires a significant amount of tuning to get the right profiles. Cycloidal functions are differentiable and easy to implement. In addition, they create motion profiles that travel fast in the middle and slower at the end points. The low acceleration moves at the end of the trajectory help to minimize the impact of the foot with the ground which can cause unexpected disturbances.

Equation (3.20) describes the swing trajectory for the transition from position $p^x_{i,k-1}$ to $p^x_{i,k}$ in the x direction for the i-th foot. Where, s is varied from 0 to 2π over the duration of the step. At the beginning of the step $s = 0$ and at the end of the step $s = 2\pi$. Equation (3.20) is also used for the y direction.

$$p^x_i(t) = p^x_{i,k-1} + (p^x_{i,k} - p^x_{i,k-1})\frac{s - sin(s)}{2\pi} \tag{3.20}$$

Equation (3.21) describes the swing trajectory for the z direction using a piece-wise function. The first function starts at $p^z_{i,k-1}$ and ends at the apex height, h. While the second function starts apex and ends at $p^z_{i,k}$. A piece-wise is required in order for the foot to be able to change elevation.

$$p_i^z(t) = \begin{cases} p_{i,k-1}^z + h\frac{1-cos(s)}{2} & s \leq \pi \\ p_{i,k}^z + (h + p_{i,k-1}^z - p_{i,k}^z)\frac{1-cos(s)}{2} & s > \pi \end{cases} \tag{3.21}$$

Figure 3.2 shows example trajectories for a swing leg that is going from 0m to 0.2m in the x direction and 0 to 0.1m in the z direction.

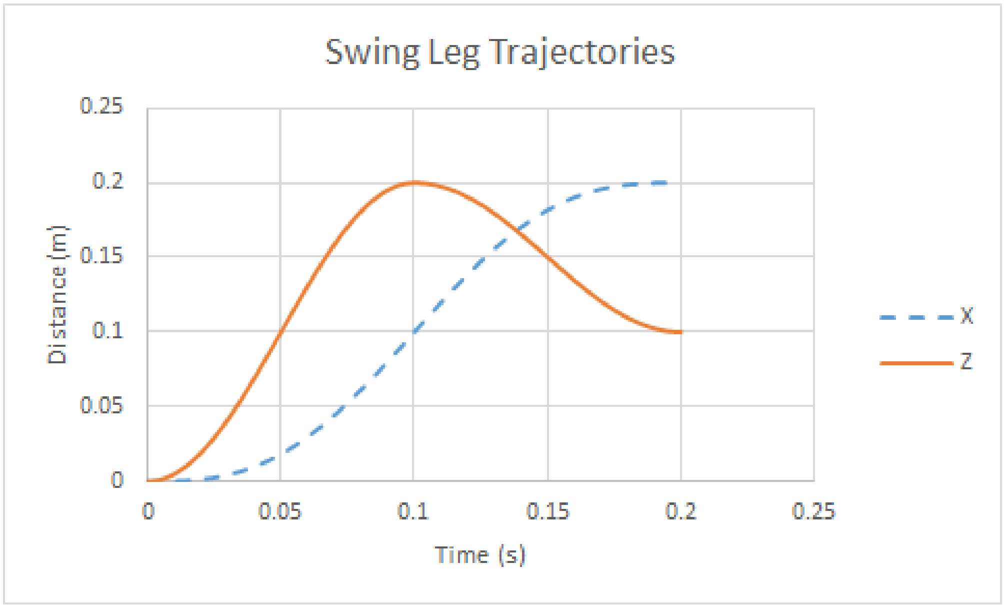


Figure 3.2: An example of a swing trajectory going from 0 to 0.2 in the x direction and 0 to 0.1 in the z direction.

3.2.4 Center of Mass Height Trajectory

To ensure that the TO problem remained convex the height of the CoM was determined apriori using cycloidal functions. If the height is added as a decision variable the ZMP dynamics constraint becomes non-linear. However, height changes of the CoM are required in order to traverse uneven terrain, therefore it was decided to pre-compute the z-trajectories based off of the coming terrain. Cycloidal functions were used because of their nice properties of continuous derivatives and small accelerations at the beginning and end. The changes of height occurred at the end of every footstep. The height was determined by taking the average position of all feet on the ground and the nominal CoM position, this is shown in (3.22) for the i-th footstep.

$$\zeta_i = r_{z,i} + \frac{C_1(t)p_1^z(t) + C_2(t)p_2^z(t) + C_3(t)p_3^z(t) + C_4(t)p_4^z(t)}{C_1(t) + C_2(t) + C_3(t) + C_4(t)} \tag{3.22}$$

Where, $r_{z,i}$ is the nominal height for the CoM at the end of the i-th footstep and C_i is the indicator function from (3.2). The trajectory for the height of the CoM is shown in (3.23).

$$r_z(t) = \zeta_{i-1} + (\zeta_i - \zeta_{i-1})\frac{s - sin(s)}{2\pi} \tag{3.23}$$

3.3 Discussion and Results

All trajectories discussed in this section were generated via PyIpopt (A python wrapper connecting to Ipopt) using an Intel Core i5/1.8GHz processesor. The duration of the CoM polynomials ($T_{k+1} - T_k$) was set to 0.05 seconds for all generated trajectories, all decision variables were initialized as 0.1, and the feet were assumed to be point feet. Position commands were sent to the actuators at 100Hz. Table 3.1 shows solve times for a range of different gaits.

3.3.1 Uneven Terrain

Fig. 3.3 shows the generated trajectory for stepping up and over a 0.14m tall cinder block which also corresponds to the last column in Table 3.1. There are three elevation changes on

Table 3.1: Solution times for different motions

	Trot	**Pace**	**Creep**	**Creep w/ 0.14m obstacle**
Distance	1.0 m	1.0 m	1.0 m	0.83 m
# of steps	8	12	20	20
Poly. δt	0.05 sec	0.05 sec	0.05 sec	0.05 sec
Solve time	0.399 sec	0.525 sec	0.833 sec	1.901 sec

the way up and three elevation changes on the way down. These represent the three different feet configurations that occurred during this trajectory: no feet on the cinder block, one foot on the cinder block, and two feet on the cinder block. The footsteps were hand tailored based off of knowledge of the terrain and robot kinematics.

3.3.2 Affects of CoM Acceleration Cost

The cost on the CoM acceleration is helpful when the desired gait has large changes in the adjacent support polygons. For example, a pacing gait with a small transition period in the middle will start on the left two legs then have a period where all four legs are on the ground followed by a stance phase of the right two legs. This will cause the support polygon to go from the left two legs to the right two legs very quickly. If only the ZMP cost was used the desired CoP would shift from the right legs then to the middle then to the left legs very quickly causing large CoM accelerations. By adding the CoM cost it was found that we could smooth out this transition from right to left by adding intermittent CoP points during

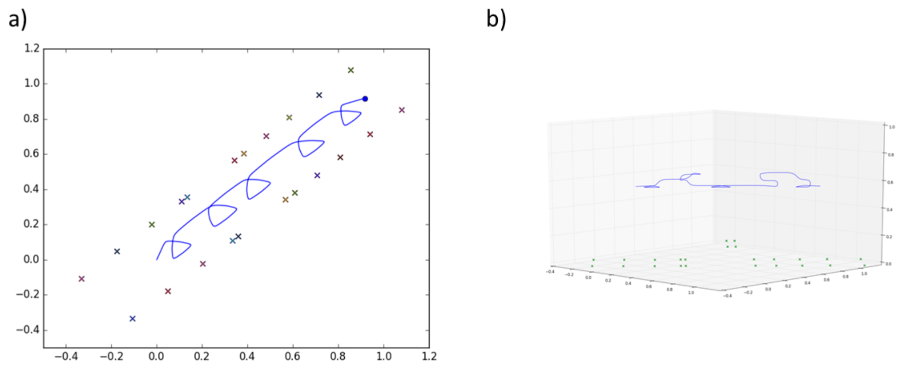


Figure 3.3: Solid blue line represents CoM trajectory and green X's represent foot positions. a) Top view of trajectory b) Side view of trajectory.

the phase where all four feet are on the ground. This phenomenon is shown in Fig. 3.4.

3.4 Center of Mass and Footstep Planning through a NLP and Vision Data

The method developed in the previous section (3.2) provided a fast trajectory optimization algorithm that can be ran online. The algorithm produced trajectories of varying time horizons but only considered the dynamics of the robot and did not consider the environment. This section builds off of that method, creating an architecture that allows for teleoperation of the robot while perceiving and reacting to the environment. An overview of the system framework for teleoperation on the ALPHRED V2 platform is shown in Fig. 3.5. User inputs (velocity magnitude, heading, and gait type) are passed to three motion planning threads, the first being the footstep planner. Based off of these user inputs the footstep planner generates footstep times and locations that will result in the desired velocity and direction

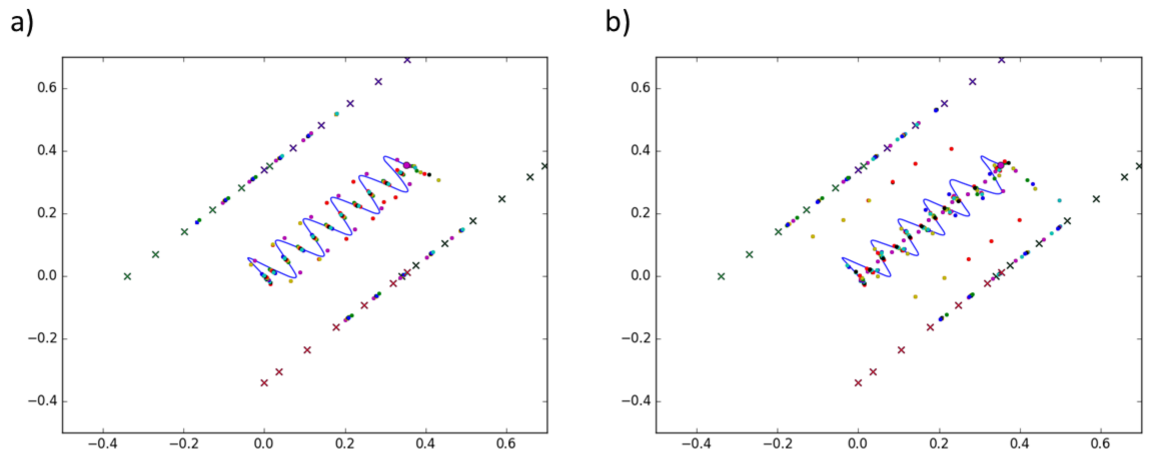


Figure 3.4: Solid blue line represents CoM trajectory, green X's represent foot positions, and colored dots represent CoP positon. a) Without CoM acceleration cost b) With CoM acceleration cost.

of the robot. The nominal footsteps are passed to the vision algorithm which potentially modifies these footsteps to the closest, viable, and safe locations. Each of these locations is given a score based off of how safe these locations are. These modified footsteps and scores are then passed to the NLP. The NLP will produce the final footstep locations and an optimal CM trajectory for the robot to execute. The optimized trajectory and footstep plan is passed to a Finite State Machine (FSM) which uses position control to execute the given trajectory. The FSM is ran at 200 Hz and is ran in parallel to the the footstep planner, vision footstep adjuster, and trajectory optimization. While the FSM is executing the given trajectory the other three threads are working on the next trajectory based off the most current user input. The three motion planning threads must be ran sequentially due to their reliance on the previous threads outcome.

3.4.1 FOOTSTEP PLANNER

The footstep planner is similar to the one discussed in Section 3.2 however it plans for the next 8 steps (2 for each foot) whereas the previous footstep planner was for an arbitrary number of steps.

$$\begin{aligned} \boldsymbol{T}_i &= [t_1, t_2, t_3, t_4] \\ \boldsymbol{F}_i &= [f_1, f_2, f_3] \end{aligned} \tag{3.24}$$

Where $i \in \{1, 2, 3, 4\}$ is the i-th leg, $t_n \in \mathbb{R}$ is a time, and $f_m \in \mathbb{R}^3$ is a 3 dimensional position vector. The entry f_1 for each leg is the current position vector of that leg. The final two position vectors are the new footstep locations that will produce the desired direction and speed of the robot. The time vector consists of a lift off time and touchdown time for each of the new footsteps respectively (no times are need for the first location).

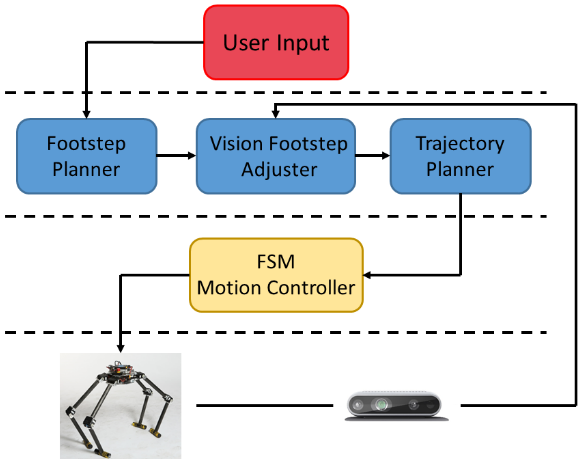


Figure 3.5: High-level system architecture flow chart. The footstep planner, vision adjuster, and trajectory planner work sequentially. While all three work in parallel to the FSM and motion controller.

Only the kinematics of the robot were considered when deciding the footstep locations of the robot. The dynamics of the robot were not a concern because these locations will be optimized later. For this reason it was relatively easy and fast to develop footstep planners for different gaits and situations.

$$f_i = f_{i,nom} + \frac{t_{i+1} - t_i}{2}\dot{r}_d \tag{3.25}$$

3.4.2 VISION FOOTSTEP ADJUSTER

Need do a little write up.

3.4.3 Trajectory Planner: Nonlinear Program

The problem formulation is similar to that of Section 3.2 however this method also optimizes for footstep locations. The addition of footstep locations makes the problem go from a convex problem to a general NLP. The notation of the NLP is the same as Section 3.2 except for the addition of the decision variables for the footstep locations. The problem is discretized into a discrete number of optimization parameters that fully describe the continuous time trajectory. The set w_c describes the CM trajectory, w_u describes the ZMP, and w_p describes the x and y location of each foot.

$$\begin{aligned} w_c &= [a_{1,0}, ..., a_{1,4}, ..., a_{n,0}, ..., a_{n,4}]^T \\ w_u &= [\lambda_{1,1}, ..., \lambda_{1,4}, ..., \lambda_{n,0}, ..., \lambda_{n,4}]^T \\ w_p &= [p_{1,1}, ..., p_{1,4}, ..., p_{m,0}, ..., p_{m,4}]^T \end{aligned} \tag{3.26}$$

Where n is the number of polynomials that describe the trajectory and m is the number of footsteps to be optimized. $a_{i,j} \in \mathbb{R}^2$ describes the i-th polynomial's j-th coefficient for both x and y, $\lambda_{i,j} \in \mathbb{R}$ describes the j-th foot's percentage of the total load at the time of the i-th polynomial, and $p_{i,j} \in \mathbb{R}^2$ describes the j-th foot's x and y position for the i-th step. Using this parameterization of the problem it becomes relatively simple to compute all required dynamics of the robot at any given time. Given,

$$\boldsymbol{A}_i = [a_{i,0}, a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4}] \quad \boldsymbol{T} = [t^0, t^1, t^2, t^3, t^4]^T \tag{3.27}$$

the CM position, velocity, and acceleration can be computed by the following:

$$\begin{aligned} \boldsymbol{r}(t) &= \boldsymbol{A}_i \boldsymbol{T} \\ \dot{\boldsymbol{r}}(t) &= \boldsymbol{A}_i \dot{\boldsymbol{T}} \\ \ddot{\boldsymbol{r}}(t) &= \boldsymbol{A}_i \ddot{\boldsymbol{T}} \end{aligned} \tag{3.28}$$

The ZMP can also be calculated with the following formulation.

$$\boldsymbol{u}(t) = \sum_{f=1}^{4} \lambda_{i,f} \boldsymbol{p}_{m,f} \tag{3.29}$$

Given that the duration of the polynomials and the step times are known it can be assumed for the remainder of this paper that the correct decision variables are chosen from the sets w_c, w_u, and w_p given a time t. Fig. 3.6 provides a broad overview of the structure of this optimization problem. Many of the costs and constraints are identical to those in Section 3.2 and for brevity are not repeated. All new costs and structuring are discussed in the subsequent sections.

$$
\begin{aligned}
\min_{w_c, w_u, w_p} \quad & \|\ddot{r}(t)\|^2 && \text{(CM acceleration)} \\
& \|\lambda(t) - \lambda^*(t)\|^2 && \text{(Optimal ZMP location)} \\
& \|p(t) - p^*(t)\|^2 && \text{(Footstep locations)} \\
\text{s.t.} \quad & && \\
& r(0) = r_0 && \text{(initial state)} \\
& r(T) = r_f && \text{(final state)} \\
& r(t_k^+) = r(t_k^-) && \text{(Continuity)} \\
& \ddot{r}(t) = f(r(t), u(t), p(t)) && \text{(ZMP dynamics)} \\
& |\boldsymbol{v}_i - r(t) + p_i(t)| > 0 && \text{(Kinematic model)} \\
& \sum \lambda_i(t) = 1 && \text{(Total Loading)} \\
& 0 < \lambda_i(t) < c_i && \text{(Single Leg Load)}
\end{aligned}
$$

Figure 3.6: An overview of the non-linear program.

3.4.3.1 Foot Placement Cost

The foot placement cost tries to minimize the distance between the location of the optimized foot location $p_{i,j}$ and the desired foot location $p^*_{i,j}$ chosen by the vision algorithm. The gain α also comes from the vision algorithm and is bound between 0 and 1. Notice that when α is 0 this cost will have no effect and the foot placement will only be chosen based off of the previously mentioned costs.

$$\theta_p \frac{10(e^{1.5\alpha} - 1)}{(e - 1)} \|p_{i,j} - p^*_{i,j}\|^2 \tag{3.30}$$

The cost was determined by trial and error in order to get the desired balance between all the other cost. Originally the foot placement cost was linear in α but this was found to either be not aggressive enough or too aggressive, it was found that an exponential function preformed much more desirably. The gain θ_p is used to change the amount of effect this cost will have on the overall optimization.

3.4.3.2 Trajectory Stitching

User inputs are checked at the beginning of every new trajectory, from the velocity commands provided by the user a nominal straight line trajectory is produced. The nominal trajectory is created simply by moving along the velocity vector given by the user input until time t_f (the final time of the optimization problem). There are two points of interest along this trajectory: the point at t_{s_1}, where t_{s_1} is the time right after all feet have finished their first step and the point at t_f. Each trajectory solves for the next 8 steps, two steps for each foot, with the desired footsteps being provided by the vision algorithm. The final CM position is set to the location of the nominal trajectory at time t_f and the velocity is set to zero. Only the first step of each foot is allowed to be a decision variable whereas the last step for each foot is set to the location provided by the vision algorithm. The final solution will create a trajectory that brings the robot to a complete stop in a nominal pose after taking two steps with each leg.

If a user input is provided, another optimization problem will be created for another 8 steps. A new inertial coordinate system is created with it's origin corresponding to the location at t_{s_1} along the previous optimization problem's nominal trajectory, as shown in Fig. 3.7. The initial conditions for the new optimization problem are the CM position, CM velocity, and footstep locations at t_{s_1}. All of these values are transformed into this new coordinate system. The previous trajectory will only execute until t_{s_1} upon which the FSM will start executing the new trajectory. This pattern will occur until no more user inputs are detected and the robot will come to a complete stop. Using a local inertial frame rather than a global inertial frame helped reduce solve times by warm starting the optimization problem with the solution from the previous optimization problem.

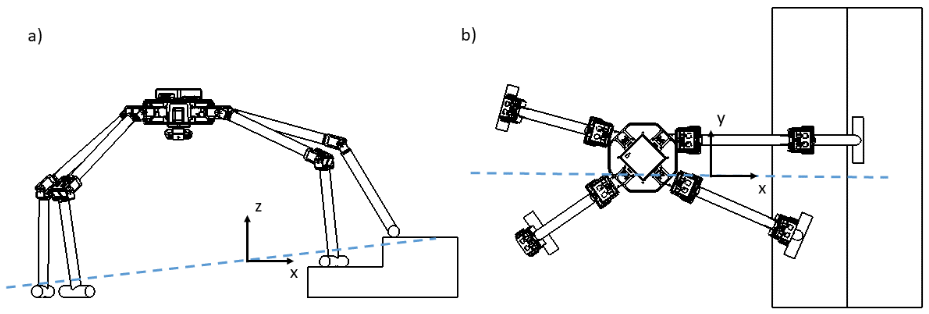


Figure 3.7: The blue dashed line represents the nominal trajectory created by the user input, a) shows the side profile of the robot, b) shows the top view.

This stitching method only uses half of the solution from each optimization problem. There are methods such as path regularization [7] that use the entire or nearly entire solution of the computed optimization problem, making these methods more efficient. We decided to use a less efficient method in order to guarantee the safety of the robot. Every trajectory that is created during our algorithm is assured to have a safe and viable stopping point meaning that if a solution to the next optimization problem fails or takes to long to solve, the robot will still be able to safely stop. Other stitching methods has no contingencies for the situation in which the optimization problem is not solved before the next trajectory is required to be executed, if this occurs the robot will freeze and most likely crash. Since non-linear programs have no guarantees on solve times we opted to use a structure that always has a safe trajectory to execute.

3.5 Discussion and Results

This section discusses the results from simulation of the ALPHRED V2 system in V-REP [21]. Unfortunately the Intel Realsense D435 was less accurate and noisier than we used for simulation. This has prevented us from implementing the algorithm in the real world. All code was written in Python 2.6 and implemented on an Intel NUC Quad-Core i7-6770HQ with 8 GB of DDR4 RAM at 2400 MHz. Interior Point Optimizer (Ipopt) [81] was used as the non-linear solver for the TO thread with PyIpopt as the Python wrapper. The simulation camera mimics the Intel Realsense D435.

In the simulation the robot trots 1.0m and then steps onto and over a 0.04m obstacle. The step length was set to 0.25m with a step time of 0.6 seconds giving the algorithm 1.2 seconds to solve before the next trajectory. During the duration of this task the algorithm stitched together eight trajectories with an average solve time of 400ms (100ms for the vision and 300ms for the TO) per trajectory. Fig. 3.8 - 3.10 are three snapshots of different moments during the simulation that best highlight the novelty of this algorithm. The figures show the nominal footsteps provided by the footstep planner (blue squares), the modified footsteps provided by the vision planner (green diamonds), and the optimized footsteps provided by the TO problem (red crosses). Fig. 3.8 is a footstep plan during the first 1.0m of travel. During this time the terrain is completely flat with no dangerous footstep locations, this is indicative of the fact that the vision algorithm did not modify the nominal footsteps and provided a low cost on all footstep. This allowed the footsteps to be highly optimized shown from the optimized footsteps deviating dramatically from the modified footsteps. In Fig. 3.9 the robot is planning on stepping onto the obstacle (dashed purple box) with the front

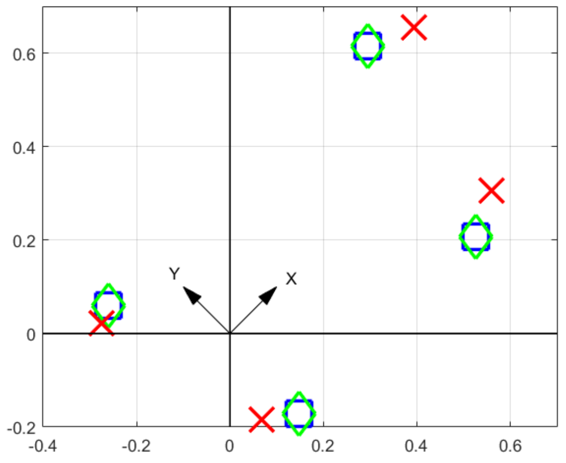


Figure 3.8: Footstep plan for flat terrain. Blue squares are nominal footsteps, green diamonds are modified footsteps, and red crosses are optimized footsteps.

two legs. The back two legs are in safe locations and are thus treated as they were in Fig. 3.8. However, the front two legs must now be modified from the nominal positions in order to provide the robot with secure footing. Due to their precarious position both the front legs have a high cost forcing the optimization algorithm to choose the position selected by the vision algorithm. An interesting result from this is that the back feet locations are now changed to try to counter act the dynamics created from modifying the front feet. This is most clearly seen by the back left's optimized footstep. The change in the back left foot's position mirrors that of the constrained front foot's location. This is a very powerful result where the optimization algorithm naturally tries to account for the adverse dynamics created by traversing safely over the obstacle. Similar phenomenon are seen in Fig. 3.10 where now the back feet are modified due to the obstacle and the front feet are free to be optimized.

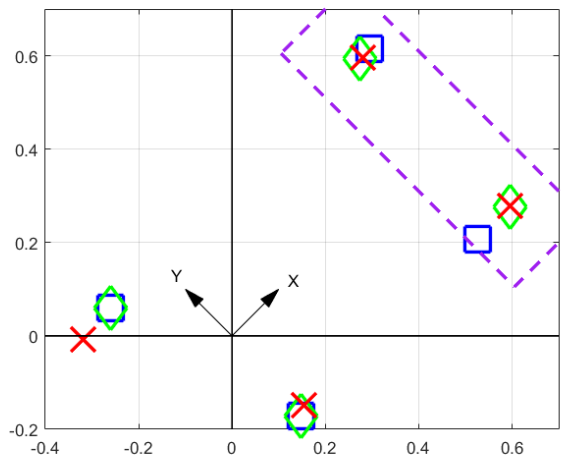


Figure 3.9: Footstep plan stepping up onto an obstacle with the front two legs. Blue squares are nominal footsteps, green diamonds are modified footsteps, and red crosses are optimized footsteps.

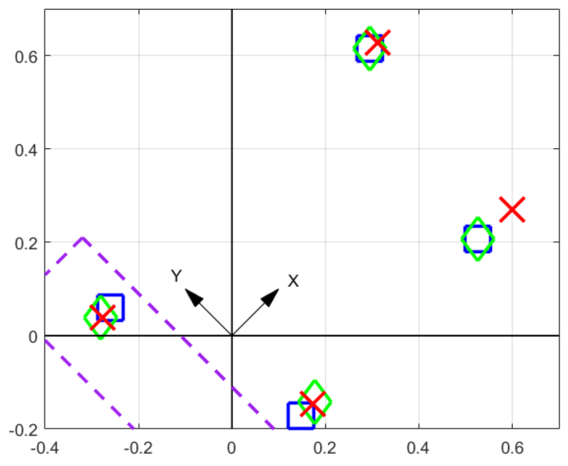


Figure 3.10: Footstep plan for stepping up onto an obstacle with the back two legs. Blue squares are nominal footsteps, green diamonds are modified footsteps, and red crosses are optimized footsteps.

CHAPTER 4

MOTION CONTROL

4.1 Limb State Machine

One of the most challenging aspects of walking robotics is the change in continuous dynamics from one state to the next. These changes occur every time a foot comes into contact with the ground or leaves the ground. Not only is it difficult to design controllers to operate in each dynamical state it can also be difficult to detect the transitions from state to state. For this reason, a Finite State Machine was developed to organize the control and detection of the changing states of each limb as shown in Fig. 4.1, similar event based strategies have been used in [13]. The transitions for the states are determined by the desired state of the limb and the actual state of the limb determined by a foot contact switch: s_1 desired to be in contact and foot switch is high, s_2 desired to not be in contact and foot switch is high, s_3 desired to be in contact and foot switch is low, and s_4 desired to not be in contact and foot switch is low. There is one more transition that fires when the end-effector velocity is too high. This transition is unique to the stance state and is described in more detail below.

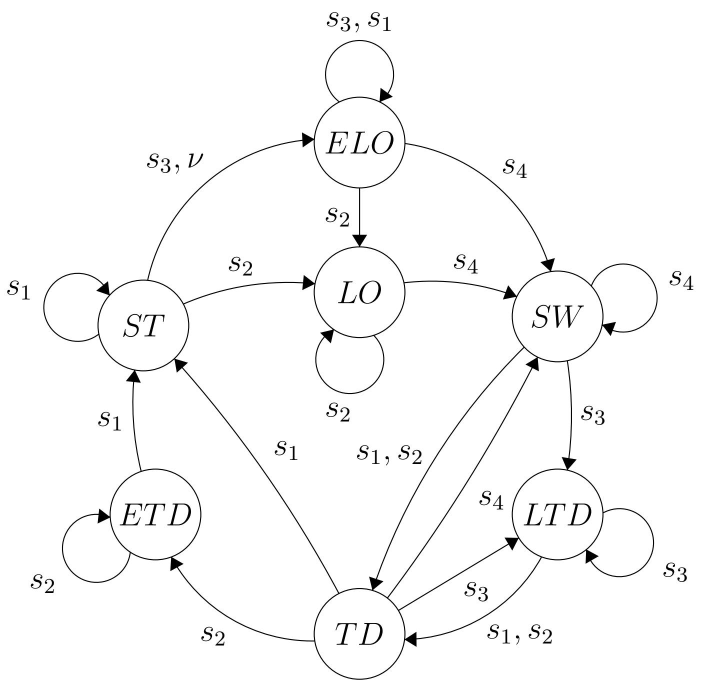


Figure 4.1: Visual representation of the limb FSM.

4.1.1 Stance (ST)

The stance phase occurs when the foot is desired to be in contact and the contact switch indicates that the foot is in contact. This state controls the dynamics of the robot to follow a desired trajectories via the ground reaction forces created by the contact. The stance state is controlled via the output of one of the locomotion controllers detailed in Section 4.2 and Section 4.3. If the controller is directly controlling the ground reaction forces at the foot via torque control as down in the controller in Section 4.3, then slipping can be a major problem. The torque controllers assume that the foot is securely on the ground and that a sufficient ground reaction force can be created however this is not the case when a foot is slipping. If a large ground reaction force is required this can cause the leg to go unstable causing damage to the robot. For this reason the stance state can transition early to the early liftoff state if the foot's velocity is calculated to be $2.5x$ the desired body velocity.

4.1.2 Swing (SW)

The swing state is used to transition the foot's position from one location to the next when both desired state is no contact and the foot switch is low. Cycloidal functions are used to create the trajectory of the swing legs [36], (4.3) describes the trajectory of the foot for the x and y body coordinates while (4.4) describes the trajectory for the z coordinate.

$$s = 2\pi \frac{T_{swing} - T}{T_{swing}} \tag{4.1}$$

$$\Delta p = p_{final} - p_{initial} \tag{4.2}$$

$$p_i^{xy} = p_{initial}^{xy} + (\Delta p^{xy}) \frac{s - sin(s)}{2\pi} \tag{4.3}$$

where, $p_i^{xy} \in \mathbb{R}^2$ is the x and y position for the i^{th} foot, T_{swing} is total duration of the swing phase, T is the remaining time in the phase which starts at T_{swing} and goes to 0 (the timing is explained in more detail in Section 3.1), $p_{initial}$ is the initial position of the foot at the beginning of the swing phase, and p_{final} is the desired final position of the foot at the end of the swing phase.

$$p_i^z(s) = \begin{cases} p_{initial}^z + max(\Delta p_z + h_{step}, h_{step}) \frac{1 - cos(s)}{2} & s < \pi \\ p_{final}^z + (p_i^z(\pi) - p_{final}^z) \frac{1 - cos(s)}{2} & s \geq \pi \end{cases} \tag{4.4}$$

During the first half of the swing phase the foot lifts to a height of h_{step} above either the initial height if the foot is stepping down or above the final height if the foot is gaining elevation. If the foot is stepping down the foot clearance is of concern, which is why the foot is commanded to lift to h_{step} above the initial height rather than the final height.

If using force control, the following control law is used to control the limb in the air:

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{K}_p\boldsymbol{M}(\boldsymbol{x}_{ref} - \boldsymbol{x}) + \boldsymbol{K}_d(\dot{\boldsymbol{x}}_{ref} - \dot{\boldsymbol{x}})) + \boldsymbol{\tau}_{ff} \tag{4.5}$$

where $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the Jacobian matrix, $\boldsymbol{K}_p \in \mathbb{R}^{3\times3}$ is the proportional gain matrix, $\boldsymbol{K}_d \in \mathbb{R}^{3\times3}$ is the derivative gain matrix, and $\boldsymbol{M} \in \mathbb{R}^{3\times3}$ is a diagonal matrix with the diagonal being equal to the diagonal of the $\boldsymbol{H}$ matrix in (4.6) and is used to scale $\boldsymbol{K}_p$ to ensure that the stiffness is the same in any configuration. The feedforward torque $\boldsymbol{\tau}_{ff}$ can be computed using the rigid body equations of motion discussed in Section 1.2.2.1:

$$\boldsymbol{\tau}_{ff} = \boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}}_{\boldsymbol{ref}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) \tag{4.6}$$

where $H \in \mathbb{R}^{3\times3}$ is the mass matrix, $C \in \mathbb{R}^{3\times3}$ is the Coriolis matrix, and $G \in \mathbb{R}^3$ is the gravitational term. Unfortunately, $\ddot{\boldsymbol{q}}_{ref}$ is not known, requiring the following substitution to be made.

$$\begin{aligned} \frac{d}{dt}\dot{\boldsymbol{x}} &= \frac{d}{dt}\boldsymbol{J}\dot{\boldsymbol{q}} && (4.7) \\ \ddot{\boldsymbol{x}} &= \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} && (4.8) \\ \ddot{\boldsymbol{q}} &= \boldsymbol{J}^{-1}(\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) && (4.9) \end{aligned}$$

Plugging (4.9) into (4.6) results in:

$$\boldsymbol{\tau}_{ff} = \boldsymbol{H}(\boldsymbol{q})\boldsymbol{J}^{-1}(\ddot{\boldsymbol{x}}_{\boldsymbol{ref}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) \tag{4.10}$$

4.1.3 Touchdown (TO)

The touchdown state is used to determine if the foot is securely on the ground. If the limb is in the swing state and a rising edge is detected from a contact sensor the limb will immediately transition into the touchdown state. Once in the touchdown state the limb will go into a low impedance mode and wait until the contact sensor reads consistently for 5ms. If the consistent reading is high then the limb will transition into one of the contact states (ST or ETD) or if the reading is low then the state will go back into the swing state until another touchdown is detected.

4.1.4 Early Touchdown (ETD)

This state performs the same function as the stance state but additionally ensures that the limb does not transition immediately to the lift-off state. This state stops the foot if it touches down before the swing state has finished and allows for an extended stance period.

4.1.5 Late Touchdown (LTO)

When this state occurs the stance controller is lacking control authority over at least one contact point, which can have a severe impact on performance. To mitigate this problem, the limb is commanded to hold it's X and Y position and move downwards in the Z direction, thus trying to shorten the time that the robot is in this state, similar methods to this were used in [32].

4.1.6 Liftoff (LO)

This state performs the same function of the swing state but additionally prevents the transition into the early touchdown until 60% of the swing time has passed. At 60% the foot is now on its downward trajectory and is capable of establishing a stable foothold, whereas this is not the case for the upward trajectory. Even if the foot runs into an obstacle on its upwards trajectory it is better to try to push through and get the foot up and over the obstacle rather than trying to establish a foothold on the side of the obstacle.

4.1.7 Early Liftoff (ELO)

Early liftoff occurs when a foot unexpectedly loses contact with the ground. When this occurs the foot stops using the output of the stance controllers and is commanded to match the inverse of the desired velocity of the robot in X and Y and to remain fixed in the Z axis. Once the limb has entered this state the it will remain in this state until the limb is desired to be in the swing phase, even if a touchdown event is detected. The motivation for this strategy is to continue to safely provide stability to the robot while an unexpected event has occurred while maintaining the desired gait. During nominal operation, a foot's velocity in the body frame should be close to the desired velocity of the body, therefore once the foot has slipped or lifted off this state will command the inverse of the desired velocity using the (4.5).

4.2 Position PD Control

During the stance and early touchdown states the X and Y Cartesian positions of the limb are commanded to match the inverse of the desired velocity however the Z position of the limb is controlled by a simple PD controller. The PD controller, described by Eq. 4.11, changes the length of the limb to correct height and rotation errors of the robot, enabling the robot to traverse uneven terrain and mitigate disturbances. In position mode the output of the PD controller is a Z velocity and in torque mode the output is a Z force; both result in changing the length of the given limb. If in torque mode, Eq. 4.5 is used to determine the desired torques.

$$\Delta z = k_{p,h}(p_{0,z} - p_z) + k_{d,h}\dot{p}_z + s(\mathbf{K_{p,r}} \log(\mathbf{R}_d\mathbf{R}^T) - \mathbf{K_{d,r}}\omega) \tag{4.11}$$

where $p_{0,z}$ is the nominal z position of the limb, $\mathbf{R}_d, \mathbf{R} \in \mathbb{R}^{3\times3}$ are the desired rotation matrix and current rotation matrix representing the transform from the body frame to the world frame (desired pitch and roll angles are always zero), $\omega \in \mathbb{R}^3$ is the angular rates of the robot in the body frame, and $s = [1, 1, 0]$ is a selection matrix that selects the roll and pitch errors (this controller does not adjust for yaw errors). The orientation errors are derived using the matrix log mapping described in [16] [56], for completeness the matrix log is derived in the following subsection. $k_{p,h}$ and $k_{d,h}$ are the scalar proportional and derivative gains for the height component. $\mathbf{K_{p,r}}$ and $\mathbf{K_{d,r}}$ are the diagonal $\mathbb{R}^{3\times3}$ proportional and derivative gain matrices for the orientation error component. For all four limbs the magnitude of the gains are the same but the orientation gain matrices will differ in signs between elements in order for the limbs to appropriately respond to errors.

4.2.1 Tracking Controller for SO(3)

One of the most widely used and effective tools for tracking a trajectory is a PD controller of the form,

$$\boldsymbol{u} = \boldsymbol{K}_p(\boldsymbol{x} - \boldsymbol{x}_d) + \boldsymbol{K}_d(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d) \tag{4.12}$$

It would be natural for one to try and use Euler angles combined with a PD controller to track a desired orientation trajectory. Unfortunately, this method does not work due to the inter-connection between the three Euler angles. Each Euler angle rotates the coordinate frame created by the Euler angle that proceeded it. A PD controller will try to fix all of the Euler angles at once however, by changing the first Euler angle this will then have effects on all proceeding Euler angles resulting in an unstable controller.

In order to overcome this deficiency Bullo *et al.* [16] derived a PD formulation that would work on the $SO(3)$ group. First, we will represent the error state as the desired frame as seen by the current frame.

$$\boldsymbol{R_e} = \boldsymbol{R}_d^T \boldsymbol{R} \tag{4.13}$$

Using the matrix exponential form of a rotation matrix we can derive the matrix logarithm from (1.36):

$$\begin{gathered} \log \boldsymbol{R}(\hat{\boldsymbol{\omega}}, \theta) = \hat{\boldsymbol{\omega}}\theta = \boldsymbol{\psi} \\ \theta = \cos^{-1}\left(\frac{1}{2}(trace(R) - 1)\right) \\ \hat{\boldsymbol{\omega}} = \frac{1}{2\sin\theta}(R - R^T) \end{gathered} \tag{4.14}$$

Notice that there is no hat symbol on $\boldsymbol{\psi}$, this implies that $\boldsymbol{\psi}$ is the $\mathbb{R}^3$ representation of $\hat{\boldsymbol{\psi}}$. Using the (4.14) on our chosen error state (4.13) we can back out the error between the axis of rotation of our current system and the axis of rotation of our desired system.

$$\boldsymbol{\psi}_e = \log(\boldsymbol{R}_e) = \log(\boldsymbol{R}_d^T \boldsymbol{R}) \tag{4.15}$$

This $\mathbb{R}^3$ representation of the orientation error is now suitable for the following stable PD controller.

$$\boldsymbol{u} = \boldsymbol{K}_p(\boldsymbol{\psi}_e) + \boldsymbol{K}_d(\boldsymbol{\omega} - \boldsymbol{\omega}_d) \tag{4.16}$$

4.2.2 Results

The primary gait used by ALPHRED is a trotting gait with swing and stance times ranging from 0.18-0.3 seconds. Using a 0.2 second swing and stance time ALPHRED can reliably walk at 1.0 m/s and achieve a turning rate of 0.5 rad/s. From this experiment the cost of transport (CoT) was calculated to be 2.6, the CoT formula used is the same as the one presented in [10]. After lowering both the swing and stance time times to 0.18 seconds ALPHRED achieved a max velocity of 1.5 m/s. Using the same trot gait, ALPHRED was able to successfully walk over an artificial obstacle course comprised of scattered blocks as obstacles ranging in height from 2-4 cm. In addition, the robot has also successfully walked around in the real world walking on side walks, streets, and dirt paths which also included terrain with slopes $> 15°$.

4.3 Model Predictive Control Using Quadratic Programming

This section presents a Model Predictive Controller (MPC) that formulates the problem into a QP to solve for the desired ground reaction forces to control the robot while in the stance and early touchdown phase. As noted through out this work one of the most challenging aspects of controlling legged systems is their discrete changes in dynamics as the points of contact change. In recent years, there have been many successful controllers that use different optimization strategies to solve for control inputs based off of the current state of the robot. For the DRC, the MIT group controlled an Atlas robot through the use of several optimization algorithms, in particular they used a NLP to simultaneously consider dynamics and kinematics as there main motion controller [54], [53]. Whereas, the IHMC DRC team used high a series of high level planners with differing time horizons to pass information to inverse dynamics QP solver [26]. They expanded this work in [32] to use a QP formulation that favors specific joint angles in the null space of the quadratic program to produce straight leg walking. Wensing et al. showed in simulation that a hierarchical QP whole-body controller could be quite powerful at producing impressive motions to control a humanoid robot [84] [83]. ETH Zurich demonstrated the use of a hierarchical whole-body controller on the ANYmal platform to produce dynamic gaits [5], [6], [7].

Though these controllers have been successful and have even created impressive forms of locomotion they all suffer from the fact that they provide an instantaneous solution that only includes the current state of the robot and does not consider the future dynamics. This results in the controller having know knowledge of predictable state changes such as a foot switching from stance to swing or periods of flight where all the feet are off the ground. In order to overcome the limitations of the previous controllers the community has started to shift their focus to MPC's as the main motion and stability controllers rather than just the high level planner. There have been a number of impressive demonstration in simulation showing the potential for MPC's as the main controller [4], [20]. One of the first successful implementations of an MPC on a real-world robot was done in [52] on the HRP-2 humanoid robot. Although, the algorithm was a success the authors acknowledged the need for improvement in computational efficiencies in order to make the controller practical for unknown environments. ETH Zurich was able to get a real-time nonlinear MPC working on the ANYmal platform through the use of a custom optimization solver [58]. This controller was able to produce a slow trot but would need to improve computational efficiency to be capable of more dynamic forms of motion. Recently, DiCarlo et al. from the Biomimetics Robotics Lab out of MIT was able to use a real-time MPC controller [19] on both Cheetah3 [12] and mini-cheetah [48] to produce a wide range of dynamic gaits. The MPC developed by DiCarlo made large simplifications of the dynamics to improve the computational efficiency. However, the simplified model was still able to retain enough of the actual dynamics to produce stable control of the robot. The work presented here is based off the MPC controller developed in [19] but is expanded to incorporate approximated swing leg dynamics and the QP is reformulated to run at +300Hz while the original ran at 50Hz.

4.3.1 Simplified Dynamics Model

The robot is modeled as a single rigid body subject to forces at the feet. These dynamics are equivalent to the floating base of whole body dynamics with massless legs. The assumption of massless legs can have drastic effects if the ratio of the mass of the body to the mass of the legs is not sufficiently high, this assumption is addressed later. Equations 4.17 and 4.18

describe the linear and rotational dynamics of the model respectively

$$\ddot{\boldsymbol{r}} = \left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{4}} \boldsymbol{f_i}/m \right) - \boldsymbol{g} \tag{4.17}$$

$$\frac{d}{dt}(\boldsymbol{I\omega}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{4}} \boldsymbol{d_i} \times \boldsymbol{f_i} \tag{4.18}$$

where, $\ddot{\boldsymbol{r}} \in \mathbb{R}^3$ is the linear acceleration of the CM, $\boldsymbol{f_i} \in \mathbb{R}^3$ is the ground reaction force at the i^{th} foot, $\boldsymbol{g} \in \mathbb{R}^3$ is the gravity vector, $\boldsymbol{I} \in \mathbb{R}^3$ is the robot's inertia tensor about the CM, $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular rate, and $\boldsymbol{d_i} \in \mathbb{R}^3$ is the vector from the CM to the i^{th} foot. All of the above variables are in the world coordinate frame.

The above equations need to be further simplified in order for the system to be written as a linear system of equations. Equation (4.18) can be further approximated by the method done in both [19] and [28]:

$$\frac{d}{dt}(\boldsymbol{I\omega}) = \boldsymbol{I\dot{\omega}} + \boldsymbol{\omega} \times (\boldsymbol{I\omega}) \approx \boldsymbol{I\dot{\omega}} \tag{4.19}$$

This approximation is only valid for systems with relatively small angular velocities where the term $\omega \times (I\omega) \approx I\dot{\omega}$ does not contribute much. Due to the fact that this controller tries to have small angular velocities the above approximation holds.

Finally, to express how the robot's orientation is progresses over time the model uses Euler angles, $\boldsymbol{\Theta} = [\phi \ \theta \ \psi]^T$, to represent the robot's orientation. The Euler angles use a Z-Y-X convention to construct a rotation matrix that represents the transform from the body frame to the world frame.

$$\boldsymbol{R_{wb}} = \boldsymbol{R_z}(\psi)\boldsymbol{R_y}(\theta)\boldsymbol{R_x}(\phi) \tag{4.20}$$

Using the convention in Eq. 4.20 the robot's angular rate can be calculated from examining how the time rate of change each Euler angle effects the system. Notice that because a Z-Y-X convention is used $\dot{\phi}$ is rotated twice, $\dot{theta}$ is rotated once, and $\dot{\psi}$ directly effects the angular rotation:

$$\boldsymbol{\omega} = \boldsymbol{R_z}(\psi)\boldsymbol{R_y}(\theta)\begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} + \boldsymbol{R_z}(\psi)\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} \tag{4.21}$$

$$\boldsymbol{\omega} = \begin{bmatrix}\cos(\theta)\cos(\psi) & \sin(\psi) & 0\\ -\cos(\theta)\sin(\psi) & \cos(\psi) & 0\\ \sin\theta & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix} \tag{4.22}$$

Taking the inverse of Eq. (4.22) leads to:

$$\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix} = \begin{bmatrix}\cos(\psi)/\cos(\theta) & -\sin(\psi)/\cos\theta & 0\\ \sin(\psi) & \cos(\psi) & 0\\ \cos\psi\tan\theta & \sin\psi\tan\theta & 1\end{bmatrix}\boldsymbol{\omega} \tag{4.23}$$

This inverse is only exists when $\theta \neq \pm\pi/2$, which is when the robot is pointed straight up and down. Fortunately, this orientation is well outside of the nominal operating point of the robot. Since the controller tries to keep the roll and pitch (ϕ and θ) at zero we can use the small angle approximation to derive the following approximation:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{\omega} = \boldsymbol{R_z}(\psi)^T \boldsymbol{\omega} \tag{4.24}$$

Combining Eq. (4.17), (4.19), and (4.24) the approximated dynamical model of the robot can be written in the following continuous time linear state space format. Notice that both $\boldsymbol{R_z(\psi)}$ and d_i are functions of the states making this a invalid representation. This issue will be addressed in the subsequent sections.

$$d/dt \begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{r} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{r}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \boldsymbol{R_z(\psi)}^T & \mathbf{0_{3\times3}} & \mathbf{0_{3\times1}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{1_{3\times3}} & \mathbf{0_{3\times1}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times1}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \boldsymbol{g} \\ \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{r} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{r}} \\ 1 \end{bmatrix} +$$
$$\begin{bmatrix} \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} \\ \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} & \mathbf{0_{3\times3}} \\ \boldsymbol{\hat{I}^{-1}}\,[\boldsymbol{d_1}]_\times & \boldsymbol{\hat{I}^{-1}}\,[\boldsymbol{d_2}]_\times & \boldsymbol{\hat{I}^{-1}}\,[\boldsymbol{d_3}]_\times & \boldsymbol{\hat{I}^{-1}}\,[\boldsymbol{d_4}]_\times \\ \mathbf{1_3}/m & \mathbf{1_3}/m & \mathbf{1_3}/m & \mathbf{1_3}/m \\ \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} & \mathbf{0_{1\times3}} \end{bmatrix} \begin{bmatrix} \boldsymbol{f_1} \\ \boldsymbol{f_2} \\ \boldsymbol{f_3} \\ \boldsymbol{f_4} \end{bmatrix} \tag{4.25}$$
$$= \boldsymbol{Ax} + \boldsymbol{Bu}$$

4.3.2 Extended State Space

The representation from (4.25) assumes that the robot has massless legs, thus applying zero forces and moments to the body during the transition from one foothold to another. This is a very common assumption in legged robotics as it greatly simplifies the dynamics allowing for simple and efficient methods to be used to create controllers [43], [42]. Using these types of controllers and the formulation above the swing leg dynamics are treated as disturbances to the model and the controller must be robust enough to counter-act those disturbances. However, if the trajectory of the swing leg is known we can predict the resultant moments and forces imposed by the swing legs from the following equations:

$$\boldsymbol{F}_{ext} = \sum_{i=1}^{N} -\boldsymbol{J}^{-T}\boldsymbol{\tau}_{ff}^{i} \tag{4.26}$$

$$\boldsymbol{M}_{ext} = \sum_{i=1}^{N} -\boldsymbol{\tau}_{ff}^{i} \tag{4.27}$$

where $\boldsymbol{\tau}_{ff}^{i}$ is the feed forward torque calculated by (4.10) for foot i. There are discrepancies between the real world leg and (4.10) which is why a PD loop is also used to control the swing leg. However, these approximations predict the general effect that the swing leg dynamics have on the body of the robot, as shown in Fig. 4.2.

Notice that in the Z direction the feedforward term dominates the overall command to the leg. In the X and Y directions the forces are relatively smaller and thus the PD and feedforward terms are sometimes of the same order of magnitude, especially at the beginning of the trajectory. The main cause of this is due to the fact that the foot has not left the ground at the beginning of the trajectory, thus the X and Y trajectories will be greatly impeded by

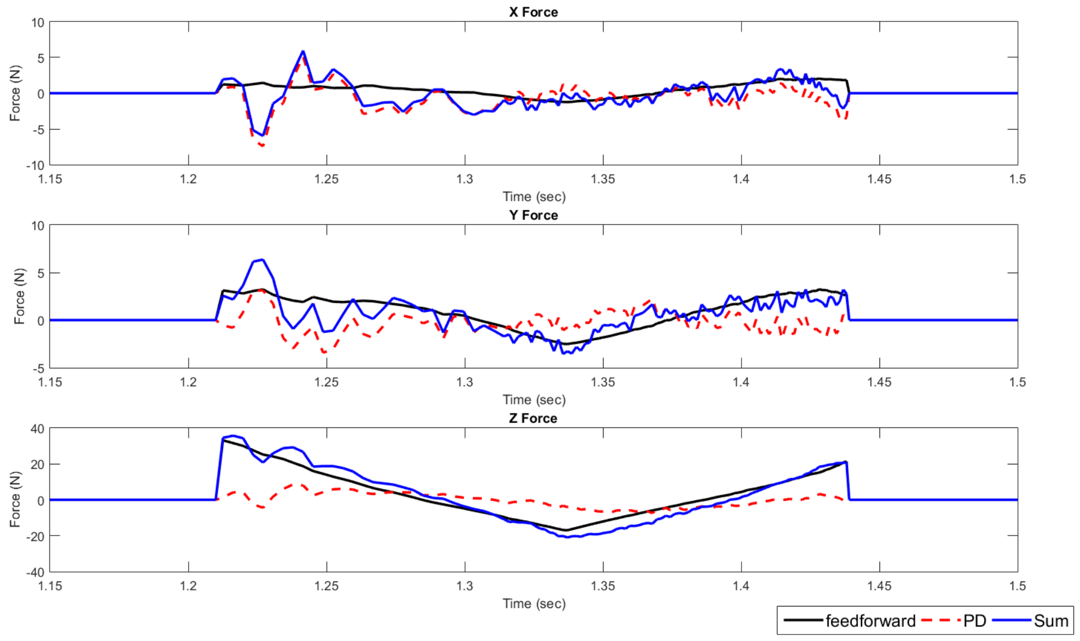


Figure 4.2: Breakdown of the contribution that the feedforward force has compared to the contribution from the PD controller during a swing leg trajectory. The solid black line represents the contribution from the feedforward term, the dashed red line represents the contribution from the PD controller, and the solid blue line represents the summation of the two which is passed to the motor controller.

a large friction force with the ground. However, for the most part the feedforward term is a good approximation of the forces that will be imparted onto the body of the robot. Equation (4.28) shows the extended version of the continuous time state space representation with the approximated swing leg dynamics.

$$
d/dt \begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{r} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{r}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{R_z}(\psi)^T & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{g} \\ \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Theta} \\ \boldsymbol{r} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{r}} \\ 1 \end{bmatrix} +
$$

$$
\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times1} \\ \hat{\boldsymbol{I}}^{-1}[\boldsymbol{d}_1]_\times & \hat{\boldsymbol{I}}^{-1}[\boldsymbol{d}_2]_\times & \hat{\boldsymbol{I}}^{-1}[\boldsymbol{d}_3]_\times & \hat{\boldsymbol{I}}^{-1}[\boldsymbol{d}_4]_\times & \hat{\boldsymbol{I}}^{-1}\boldsymbol{M_{ext}} \\ \mathbf{1}_3/m & \mathbf{1}_3/m & \mathbf{1}_3/m & \mathbf{1}_3/m & \boldsymbol{F_{ext}}/m \\ \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \\ \boldsymbol{f}_4 \\ 1 \end{bmatrix} \tag{4.28}
$$

$$
= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}_{ext}\boldsymbol{u}
$$

4.3.3 Discrete Time Model

The continuous time linear model can be represented as the following extended continuous time model, (4.29).

$$
d/dt \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}_{ext} \\ 0 & 0 \end{bmatrix}}_{\boldsymbol{M}} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} \tag{4.29}
$$

Using the state transition matrix the discritized extended state space model can be computed using the following matrix exponential.

$$\hat{\boldsymbol{M}} = e^{MT} \tag{4.30}$$

Where T is the time step for the discrete time model. The matrix exponential can be expressed as an infinite series.

$$e^{MT} = \boldsymbol{I} + \boldsymbol{M}T + \frac{1}{2!}\boldsymbol{M}^2T^2 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i!}\boldsymbol{M}^iT^i \tag{4.31}$$

Upon further inspection of M we see that M is upper triangular resulting in $\boldsymbol{M}^3 = \boldsymbol{0}$ thus,

$$e^{MT} = \boldsymbol{I} + \boldsymbol{M}T + \frac{1}{2!}\boldsymbol{M}^2T^2 \tag{4.32}$$

which can be efficiently computed. Unpacking $\hat{\boldsymbol{M}}$ we recover the discretized version of our original state space model.

$$\hat{\boldsymbol{M}} = \begin{bmatrix} \hat{\boldsymbol{A}} & \hat{\boldsymbol{B}}_{ext} \\ 0 & 0 \end{bmatrix} \tag{4.33}$$

Finally we can express the dynamics of our system as the following discrete time state space model.

$$\hat{\boldsymbol{x}}[k+1] = \hat{\boldsymbol{A}}\boldsymbol{x}[k] + \hat{\boldsymbol{B}}_{ext}[k]\boldsymbol{u}[k] \tag{4.34}$$

Notice that $\hat{\boldsymbol{A}}$ is a constant however in (4.28) $\boldsymbol{A}$ is a function of the state ψ. To write $\hat{\boldsymbol{A}}$ as a constant, ψ is approximated as the average yaw angle across the entire trajectory. In addition, $\hat{\boldsymbol{B}}_{ext}[k]$ are all functions of $\boldsymbol{d}_i$ the distance from the CM to the foot location at timestep k. This is also a function of the state, however if $\hat{\boldsymbol{B}}_{ext}[0]$ uses the current state of the robot and all future CM and foot locations are assumed to follow the desired trajectory perfectly then all $\boldsymbol{d}_i$'s can be known apriori allowing for the calculation of all $\hat{\boldsymbol{B}}_{ext}[k]$ across the entire time horizon.

Both of these assumptions are large approximations and only accurate if the robot is able to follow the desired trajectory. Unfortunately large disturbances can cause the robot to deviate substantially from the desired trajectory making the approximation of $\hat{\boldsymbol{B}}_{ext}[k]$ inaccurate. However, the model is exact for the first time step because the initial state of the robot is used to create the discrete time model. And due to the fact that this is being constructed as an MPC only the solution for the first timestep will be used before the problem is reconstructed and solved again using the current state of the robot.

4.3.4 Quadratic Programming Formulation

The MPC is formulated as a quadratic program in the following form (see Section 1.2.1.3 for more details on quadratic programming).

$$\begin{aligned} \min_{z} \quad & 1/2 z^T M z + c^T z \\ \text{s.t.} \quad & \\ & Az \leq b \end{aligned} \tag{4.35}$$

The problem is discretized over a finite horizon using the dynamics derived in the previous section to propagate the state of the robot into the future. There are different ways of formulating the problem as a quadratic program, the following sections talk about a direct transcription method and an indirect transcription method. The optimization variables z will change depending on which method is used. However, in both cases z will include the ground reaction forces for each leg at each discretized point in time. In addition, each formulation will contain a set of constraints that ensures the ground reaction forces remain inside the friction cone.

$$\begin{aligned} f^i_{x,k} - \mu f^i_{z,k} &\leq 0 \\ -f^i_{x,k} - \mu f^i_{z,k} &\leq 0 \\ f^i_{y,k} - \mu f^i_{z,k} &\leq 0 \\ -f^i_{y,k} - \mu f^i_{z,k} &\leq 0 \end{aligned} \tag{4.36}$$

Where $f^i_{x,k}$ is the ground reaction force in the x direction for the i^{th} at time step k. This constraint creates a friction pyramid that is contained inside of the friction cone ensuring that the foot does not slip. The friction coefficient μ must be decided upon based on knowledge of the ground that the robot will be operating on.

4.3.4.1 Indirect Method

The indirect method only optimizes over the ground reaction forces at each foot.

$$\boldsymbol{z} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_N]^T = [\boldsymbol{f}^1_0, \boldsymbol{f}^2_0, \boldsymbol{f}^3_0, \boldsymbol{f}^4_0, \boldsymbol{f}^1_1, \boldsymbol{f}^2_2, \boldsymbol{f}^3_3, \boldsymbol{f}^4_4, \cdots, \boldsymbol{f}^1_N, \boldsymbol{f}^2_N, \boldsymbol{f}^3_N, \boldsymbol{f}^4_N]^T \tag{4.37}$$

Where $\boldsymbol{u}_k$ is the vector of all the ground reaction forces for all of the feet at time step k, and $\boldsymbol{f}_k^i$ is the ground reaction forces for the i^{th} foot at time step k. If all of the ground reaction forces are known and the initial state of the robot is known then the future states of the robot can be calculated. Starting from the initial state x_0 and using the dynamics from (4.34) the state of the robot at $k = 1$ can be calculated, (4.38). Once again using the dynamics from (4.34) the state of the robot at $k = 2$ can be calculated, (4.39). However, if we plug in the solution for the state of the robot at (4.38) for $\boldsymbol{x}_1$ then the state $\boldsymbol{x}_2$ can be written as a function of the initial state and the known control inputs as shown in (4.41). This method can be repeated for each time step, recursively solving for each state as a function of the initial state and the control inputs.

$$\begin{aligned}
\boldsymbol{x}_1 =& \hat{\boldsymbol{A}}\boldsymbol{x}_0 + \hat{\boldsymbol{B}}_0\boldsymbol{u}_0 && (4.38)\\
\boldsymbol{x}_2 =& \hat{\boldsymbol{A}}\boldsymbol{x}_1 + \hat{\boldsymbol{B}}_1\boldsymbol{u}_1 && (4.39)\\
=& \hat{\boldsymbol{A}}(\hat{\boldsymbol{A}}\boldsymbol{x}_0 + \hat{\boldsymbol{B}}_0\boldsymbol{u}_0) + \hat{\boldsymbol{B}}_1\boldsymbol{u}_1 && (4.40)\\
=& \hat{\boldsymbol{A}}^2\boldsymbol{x}_0 + \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}_0\boldsymbol{u}_0 + \hat{\boldsymbol{B}}_1\boldsymbol{u}_1 && (4.41)\\
\boldsymbol{x}_3 =& \hat{\boldsymbol{A}}\boldsymbol{x}_2 + \hat{\boldsymbol{B}}_2\boldsymbol{u}_2 && (4.42)\\
=& \hat{\boldsymbol{A}}(\hat{\boldsymbol{A}}^2\boldsymbol{x}_0 + \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}_0\boldsymbol{u}_0 + \hat{\boldsymbol{B}}_1\boldsymbol{u}_1) + \hat{\boldsymbol{B}}_2\boldsymbol{u}_2 && (4.43)\\
=& \hat{\boldsymbol{A}}^3\boldsymbol{x}_0 + \hat{\boldsymbol{A}}^2\hat{\boldsymbol{B}}_0\boldsymbol{u}_0 + \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}_1\boldsymbol{u}_1 + \hat{\boldsymbol{B}}_2\boldsymbol{u}_2 && (4.44)\\
&\vdots && (4.45)\\
\boldsymbol{x}_N =& \hat{\boldsymbol{A}}^N\boldsymbol{x}_0 + \hat{\boldsymbol{A}}^{N-1}\hat{\boldsymbol{B}}_0\boldsymbol{u}_0 + \cdots + \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}_{N-1}\boldsymbol{u}_{N-1} + \hat{\boldsymbol{B}}_N\boldsymbol{u}_N && (4.46)
\end{aligned}$$

Collecting all the the terms multiplied by the initial state $\boldsymbol{x}_0$ into a matrix $\boldsymbol{A}_{qp} \in \mathbb{R}^{13N\times 13}$ and all the terms multiplied by the control inputs into a matrix $\boldsymbol{B}_{qp} \in \mathbb{R}^{13N\times 12N}$ the entire system can be written into a single system of equations (4.47) with $\boldsymbol{X} \in \mathbb{R}^{13N}$ representing the state of the robot at each point in time across the entire time horizon. This representation is similar to the condensed formulation discussed in [40].

$$\boldsymbol{X} = \boldsymbol{A}_{qp}\boldsymbol{x}_0 + \boldsymbol{B}_{qp}\boldsymbol{z} \tag{4.47}$$

$$\boldsymbol{A}_{qp} = \begin{bmatrix} \hat{\boldsymbol{A}} \\ \hat{\boldsymbol{A}}^2 \\ \\ \vdots \\ \\ \hat{\boldsymbol{A}}^{N-1} \\ \hat{\boldsymbol{A}}^{N} \end{bmatrix} \quad \boldsymbol{B}_{qp} = \begin{bmatrix} \hat{\boldsymbol{B}}_0 & 0 & & \cdots & 0 \\ \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}_0 & \hat{\boldsymbol{B}}_1 & 0 & & \\ & & & & \\ \vdots & & & \ddots & \vdots \\ & & & & \\ & & & & 0 \\ \hat{\boldsymbol{A}}^{N-1}\hat{\boldsymbol{B}}_0 & \hat{\boldsymbol{A}}^{N-2}\hat{\boldsymbol{B}}_1 & \hat{\boldsymbol{A}}^{N-3}\hat{\boldsymbol{B}}_2 & \cdots & \hat{\boldsymbol{B}}_N \end{bmatrix} \tag{4.48}$$

A cost function was chosen that is common in reference tracking problems, which minimizes the weighted least-squares error from the reference trajectory and the overall control effort:

$$\boldsymbol{J} = ||\boldsymbol{X} - \boldsymbol{X}_{ref}||_Q + ||\boldsymbol{z}||_R \tag{4.49}$$

$$= ||\boldsymbol{A}_{qp}\boldsymbol{x}_0 + \boldsymbol{B}_{qp}\boldsymbol{z} - \boldsymbol{X}_{ref}||_Q + ||\boldsymbol{z}||_R \tag{4.50}$$

Where $\boldsymbol{Q} \in \mathbb{R}^{13N\times 13N}$ is the weighting matrix for the states of the robot and $\boldsymbol{R} \in \mathbb{R}^{12N\times 12N}$

is the weighting matrix for the control effort. Multiplying out the cost function results in:

$$\boldsymbol{J} = \boldsymbol{z}^T(\boldsymbol{B}_{qp}^T\boldsymbol{Q}\boldsymbol{B}_{qp} + \boldsymbol{R})\boldsymbol{z} + \tag{4.51}$$

$$\boldsymbol{z}^T 2\boldsymbol{B}_{qp}^T\boldsymbol{R}(\boldsymbol{A}_{qp}\boldsymbol{x}_0 - \boldsymbol{X}_{ref}) + \tag{4.52}$$

$$\underbrace{\boldsymbol{x}_0^T\boldsymbol{A}_{qp}^T\boldsymbol{R}\boldsymbol{A}_{qp}\boldsymbol{x}_0 + \boldsymbol{X}_{ref}^T\boldsymbol{R}\boldsymbol{X}_{ref} - \boldsymbol{x}_0^T\boldsymbol{A}_{qp}^T\boldsymbol{R}\boldsymbol{X}_{ref}}_{\text{Constant with respect to decision variables}} \tag{4.53}$$

Finally we can construct the problem into the standard quadratic programming form (4.35), note that the terms that are constant with respect to the optimization variables can be ignored.

$$\boldsymbol{M} = 2(\boldsymbol{B}_{qp}^T\boldsymbol{Q}\boldsymbol{B}_{qp} + \boldsymbol{R}) \tag{4.54}$$

$$\boldsymbol{c} = 2\boldsymbol{B}_{qp}^T\boldsymbol{R}(\boldsymbol{A}_{qp}\boldsymbol{x}_0 - \boldsymbol{X}_{ref}) \tag{4.55}$$

The quadratic cost $\boldsymbol{M}$ results in a dense matrix of the size $12N \times 12N$, and the linear cost $\boldsymbol{c}$ results in a vector of the size $12Nx1$. The condensed formulation used to construct the problem ensures that the dyanmics are always satisfied. For this reason the only constraints for the problem are the friction cone constraints discussed above. Typically the problem is discretized into 10-15 time steps resulting in a relatively small quadratic program that can be solved in milliseconds.

4.3.4.2 Direct Method

The direct method optimizes over the state of the robot and the ground reaction forces at each foot.

$$\boldsymbol{z} = [\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{u}_N]^T \tag{4.56}$$

The same cost function as the indirect method is used but now with the state vectors being added to the decision variables the representation of the cost function becomes trivial.

$$\boldsymbol{J} = \sum_{k=0}^{N} ||\boldsymbol{x}_k - \boldsymbol{x}_{k,ref}||_L + ||\boldsymbol{u}_k||_P \tag{4.57}$$

Where $\boldsymbol{L} \in \mathbb{R}^{13\times 13}$ is the weighting matrix for the states of the robot and $\boldsymbol{P} \in \mathbb{R}^{12\times 12}$ is the weighting matrix for the control effort. Multiplying out the cost function results in:

$$\boldsymbol{J} = \sum_{k=0}^{N} \boldsymbol{x}_k^T \boldsymbol{L} \boldsymbol{x}_k - 2\boldsymbol{x}_k^T \boldsymbol{L} \boldsymbol{x}_{k,ref} + \boldsymbol{u}_k^T \boldsymbol{P} \boldsymbol{u}_k + \boldsymbol{x}_{ref}^T \boldsymbol{L} \boldsymbol{x}_{ref} \tag{4.58}$$

$$= \sum_{k=0}^{N} \begin{bmatrix} \boldsymbol{x}_k^T & \boldsymbol{u}_k^T \end{bmatrix} \begin{bmatrix} \boldsymbol{L} & 0 \\ 0 & \boldsymbol{P} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \end{bmatrix} + \begin{bmatrix} \boldsymbol{x}_k^T & \boldsymbol{u}_k^T \end{bmatrix} \begin{bmatrix} -2\boldsymbol{L}\boldsymbol{x}_{k,ref} \\ 0 \end{bmatrix} + \boldsymbol{x}_{ref}^T \boldsymbol{L} \boldsymbol{x}_{ref} \tag{4.59}$$

$$= \sum_{k=0}^{N} \boldsymbol{z}_k^T \begin{bmatrix} \boldsymbol{L} & 0 \\ 0 & \boldsymbol{P} \end{bmatrix} \boldsymbol{z}_k + \boldsymbol{z}_k^T \begin{bmatrix} -2\boldsymbol{L}\boldsymbol{x}_{k,ref} \\ 0 \end{bmatrix} + \boldsymbol{x}_{ref}^T \boldsymbol{L} \boldsymbol{x}_{ref} \tag{4.60}$$

Once again we can formulate the problem into the standard QP form (4.35) by using the following matrices as the quadratic cost and linear cost.

$$\boldsymbol{M} = 2 \begin{bmatrix} \boldsymbol{L} & 0 & \cdots & & 0 \\ 0 & \boldsymbol{P} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \boldsymbol{L} & \\ 0 & & \cdots & & \boldsymbol{P} \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} -2\boldsymbol{L}\boldsymbol{x}_{0,ref} \\ 0 \\ \vdots \\ -2\boldsymbol{L}\boldsymbol{x}_{N,ref} \\ 0 \end{bmatrix} \tag{4.61}$$

In this case the quadratic cost matrix M is now of size $25N \times 25N$ and the linear cost vector is of size $25N \times 1$. In addition, the dynamics are not naturally encoded into the problem, therefore constraints need to be added to the problem in addition to the friction constraints from equation (4.36). An equality constraint can be used to ensure that the dynamics of the system are satisfied:

$$\boldsymbol{x}_{k+1} - \hat{\boldsymbol{A}}\boldsymbol{x}_k - \hat{\boldsymbol{B}}_k\boldsymbol{u}_k = 0 \tag{4.62}$$

4.3.4.3 Comparison of the Direct Versus Indirect Method

Running real time optimization algorithms on a robot is challenging and requires fast solutions that are guaranteed to converge. Both the direct and indirect methods formulate the problem into a QP which guarantees convergence. The speed at which the problem is solved is another important aspect, if the solution isn't found at a fast enough rate the robot will become unstable and the controller will fail. Most researchers focus on how fast the optimization takes once the problem has been formulated and passed to a given QP solver, however another aspect of the optimization problem is how long does it take to formulate the problem before passing it to the solver.

Looking first at the solve times of the QP solvers we see that both formulations are $\mathcal{O}(N^2)$, with N being the number of decision variables. This means that the problem sizes increase equally in complexity as the number of decision variables increases. However, the direct method has twice as many decision variables per time step compared to that of the indirect method. This would suggest that the indirect method would calculate solutions significantly faster than the direct method. However, if we look closer at how the direct method is formulated we see that the Hessian matrix is a diagonal matrix and that the constraint Jacobian is a block diagonal matrix. The structure of these matrices help to greatly reduce the time required to solve the problem. Whereas, with the indirect method the Hessian might be significantly smaller but it is a dense matrix which can not be exploited to reduce the solution times of the problem. For this reason we see that the solution times for the direct method are less than 1ms slower than the indirect method, Fig. 4.3.

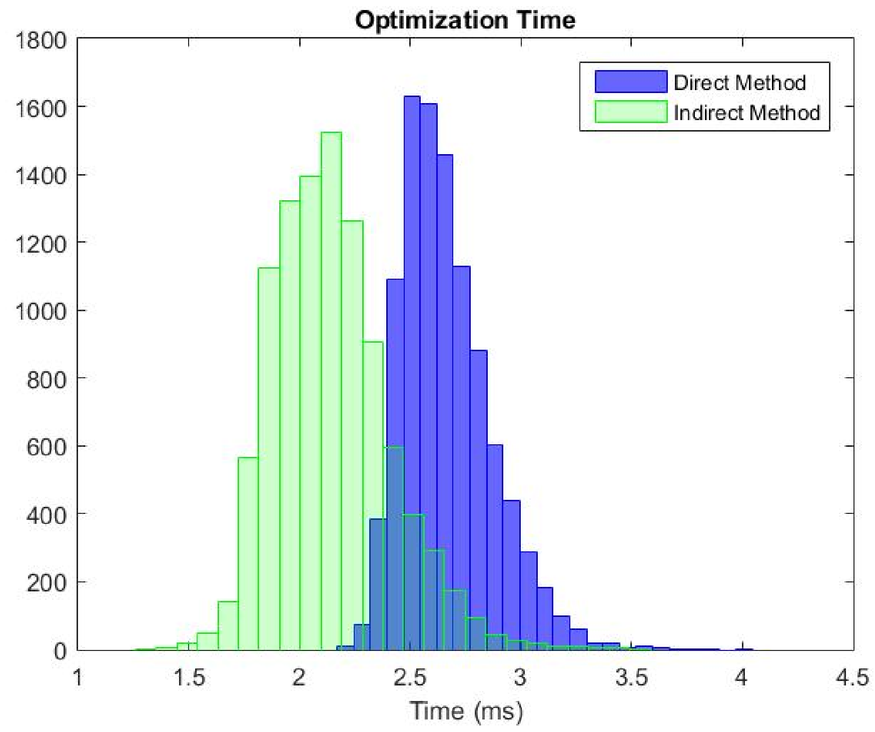


Figure 4.3: Comparison of time required to solve the QP problem for the direct method versus indirect method over 10000 samples.

The indirect method has a slight advantage in solution times over the direct method but this is not the case for the time it takes to formulate the QP problem. Looking first at the indirect method, for every solution the A_{eq} dense $N \times 13$ and the B_{eq} lower triangular $N \times N$ matrices must be formulated. From these matrices the dense $N \times N$ Hessian matrix and the dense $N \times 1$ linear cost vector need to be constructed for every time step. The constraint Jacobian only needs to be created for the first solution as the only constraints are the friction constraints and these do not change based off of the state of the robot. Looking at the number of calculations required for the direct method we see that the same $N \times 1$ linear cost matrix created in the indirect method also has to be created for the direct method. However, the $N \times N$ Hessian matrix for direct method is not only diagonal but it is also constant, which means this matrix only has to be created once. The constraint Jacobian is block diagonal with the same constant friction constraint but with time varying dynamic constraints. Therefore, n_p constraints must be updated at every time step (which is approx. 10-15) in the constraint Jacobian. As a result the indirect method is $\mathcal{O}(N^2)$ in the number of computations required to formulate the problem whereas the direct method is only $\mathcal{O}(N)$, which is most readily seen in Fig. 4.4 comparing the time required to formulate the problem for the two methods.

Combining the results from the solution times and the formulation times Fig. 4.5 shows the time required to run the MPC controller using the indirect method and the direct method. Considering that the direct method was only slightly slower for solution times but far superior in formulation times we see that the direct method on average takes 3.2ms to run whereas the indirect method takes 7.1ms to run resulting in a $> 2\times$ improvement from the indirect method used in [19].

All time tests were run on Razer Blade laptop with an Intel Core i7-7700HQ 2.80 GHz CPU and 16.0 GB of RAM. All code was written in C++ using the Eigen3 matrix library [35]. Gurobi 8.1.1 [59] was used as the QP solver. The code was compiled using the -0fast option for gcc compilers which gave a large efficiency boost to the overall computation times for both methods. The exact same series of robot states were used for both methods to

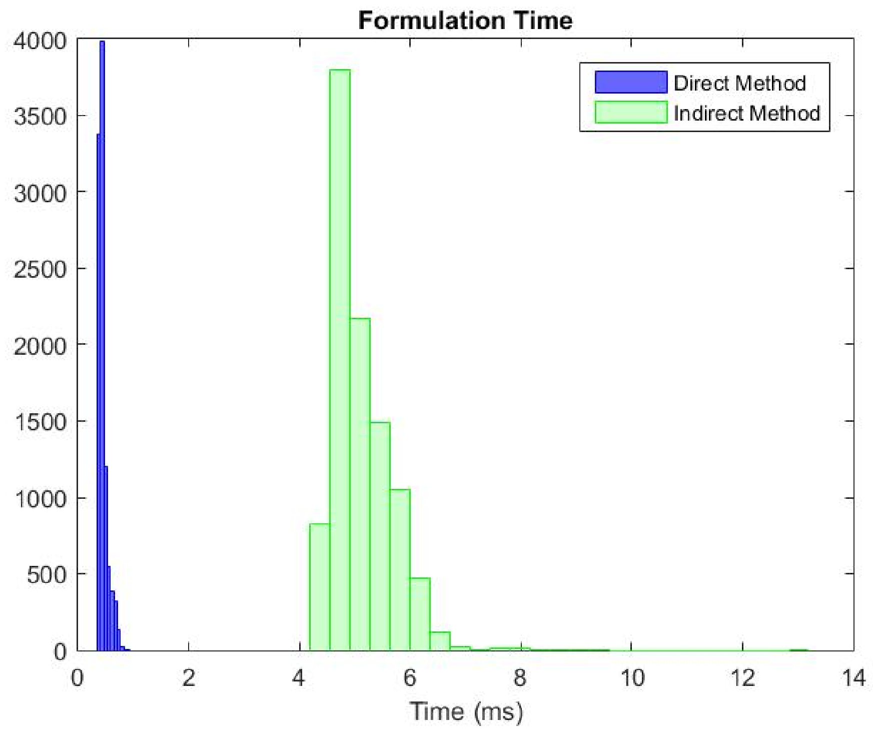


Figure 4.4: Comparison of time required to formulate the QP problem for the direct method versus indirect method over 10000 samples.

compile the 10000 data points recorded during testing.

4.3.5 Implementation Techniques

The process of transferring a controller from simulation to the real-world system is often difficult due to the non-linearities of the real world. These challenges often require solutions that require unique insight into the problem and can by system dependent. This section outlines some of the challenges and solutions of implementing the MPC controller on the ALPHRED 2 platform.

The first challenge came from the method with which the MPC similifies the inertia of

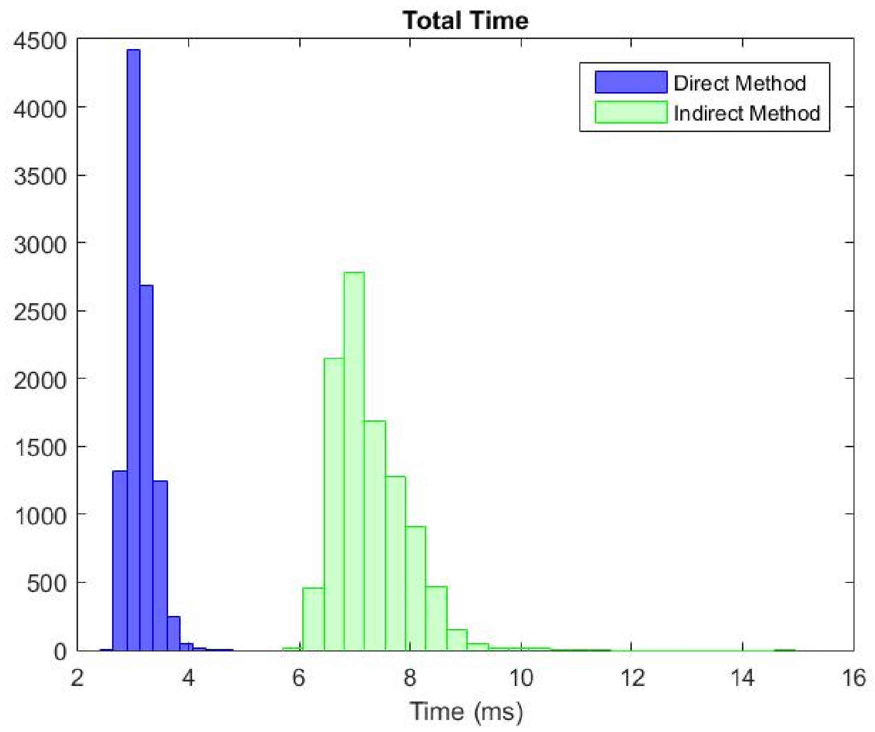


Figure 4.5: Comparison of total cycle time for the controllers using the direct method versus indirect method over 10000 samples.

the entire robotic platform. In order to write down the dynamics of the robot into a discete time linear state space model the inertia of the robot was approximated to be the inertia of the entire robot in it's nominal pose. The inertia tensor that resulted from this is the following:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} 0.85 & 0 & 0 \\ 0 & 1.30 & 0 \\ 0 & 0 & 1.13 \end{bmatrix} \tag{4.63}$$

The above inertia tensor is in accordance with how the approximation was derived. Considering that the nominal pose is elongated parallel to the sagittal plane it would be expected that the I_{yy} and I_{zz} components of the inertia tensor are larger than the I_{xx} component, and

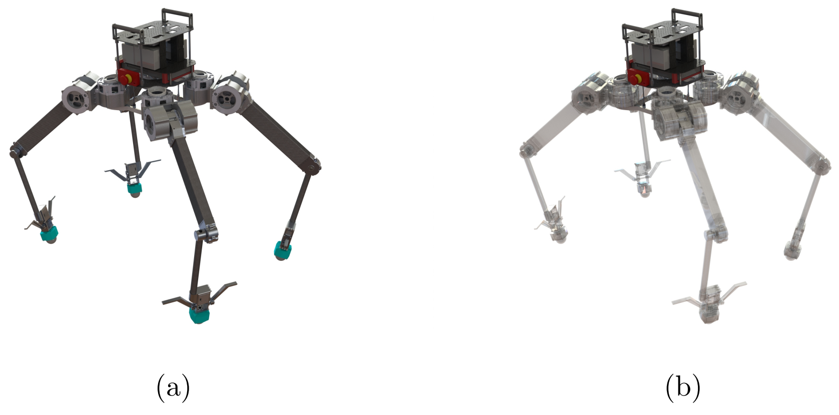


Figure 4.6: a) Pose of the robot that was used to approximate the inertia tensor assuming the entire robot as a single rigid body. b) The solid part representing the actual inertia that should be considered for the rotation about the Z axis.

this would be correct if the entire system was one rigid body. However, the system is not a single rigid body thus additional care must be used to analyze the system. Analyzing how the robotic system would have to move to perform a pure rotation about a given axis we can analyze how accurate the Inertia tensor approximation really is. Focusing on the rotation about the X and Y axes we see that for both of these rotations complex multi-axis solutions are required to perform a pure rotation about these axes. This means that the entire robot must move to produce this type of movement and thus the approximations used for these axes is valid. However, the rotation about the Z axis does not require multi-axis coordinated moves, this rotation only requires actuation from the hip yaw actuators. To perform a pure yaw only the light weight body moves, which is much less inertia then the original approximation (Fig. 4.6). Using the original I_{zz} in (4.63) resulted in excesses overshooting in the yaw create undesirable oscillations and instability.

In order to mitigate the problem a combination of solutions were implemented. The first being reducing the I_{zz} term to the average of the inertia of the entire rigid body system and the inertia of just the body. The inertia of just the body is the actual inertia when all four limbs are on the ground however as limbs leave the ground the yaw inertia increases until the limit is reached and no limbs are on the ground and the yaw inertia is equal to the original I_{zz}. For this reason the average between the two inertia's is used as the approximated yaw inertia in the controller. Unfortunately, because the discrepancy between the yaw inertia when all the feet are on the ground and when no feet are on the ground is so large the average between the two still results in a controller that is too aggressive in the yaw axis. In a traditional PD controller the solution would be to increase the derivative gain resulting in a damping effect. Unfortunately, with this whole body controller there is no one knob to turn to provide a damping effect. Increasing or decreasing the gains on the yaw states of the MPC changed the range of instability of the controller but it did not result in a fully stable controller. To provide a dampening effect the desired trajectory was changed to be less aggressive of a trajectory. The desired trajectory of the robot is created purely from the desired linear velocities and yaw rate provided by the user. This method tells the controller that it is desired for the controller to match the desired trajectory in a single time step if possible. To simulate a dampening effect the desired yaw rate was calculated by blending the current yaw rate and the desired yaw rate (4.64) which now tells the controller to fix the yaw error over a number of time steps rather than a single time step.

$$\dot{\psi}_{desired} = (1 - \beta^k)\dot{\psi}_{user} + \beta^k \dot{\psi}_{actual} \tag{4.64}$$

Where β is a parameter between 0 and 1 used to control how gradual the transition from the actual state to the user desired state is in the trajectory passed to the MPC. The smaller the value of β the more aggressive the desired trajectory will be. In practice a value of ~ 0.7 was used for β. The combination of lowering the yaw inertia value and smoothing the desired trajectory stabilized the control of the yaw axis.

The next major challenge was due to the difficult dynamics of touchdown sensing and control. In the ideal situation the controller would immediately detect when a foot touches the ground enabling the controller to instantaneously stop the foot and switch to the stance phase creating a perfect inelastic collision. However, in practice there are time delays in sensing when the foot hits the ground and there are imperfections of the control of the swing leg causing the foot to slam into the ground resulting in a bounce.

The timing delays come from the analog to digital conversion of the foot switch and the non-negligible actuation force required to trigger the foot switch. The foot consists of a rubber half-sphere with a hardened epoxy core. There is no rigid coupling between the foot and the tibia of the robot, but instead the foot is floating on a thin rubber case as indicated in Fig. 4.7. The rubber allows the foot to move a small amount in all directions until the rubber has been compressed enough for the foot to make a rigid contact with the tibia. Ground contact is detected by using analog on/off switches to determine if the rubber has been compressed and the foot has moved. The activation force required to compress the rubber can change based off of the amount of rubber used or the type of rubber used. Ideally, the foot would have a low activation force so that touchdown could be detected immediately. Unfortunately if too low of an activation force is used then the foot will trigger during the swing phase of walking. The foot itself has a non-negligible mass and as the swing phase duration decreases the acceleration profile of the foot increases resulting in a force applied to the rubber which prematurely triggers the activation of the ground detection switches. For this reason, the activation force is ~ 10 N to prevent false positives during the swing phase.

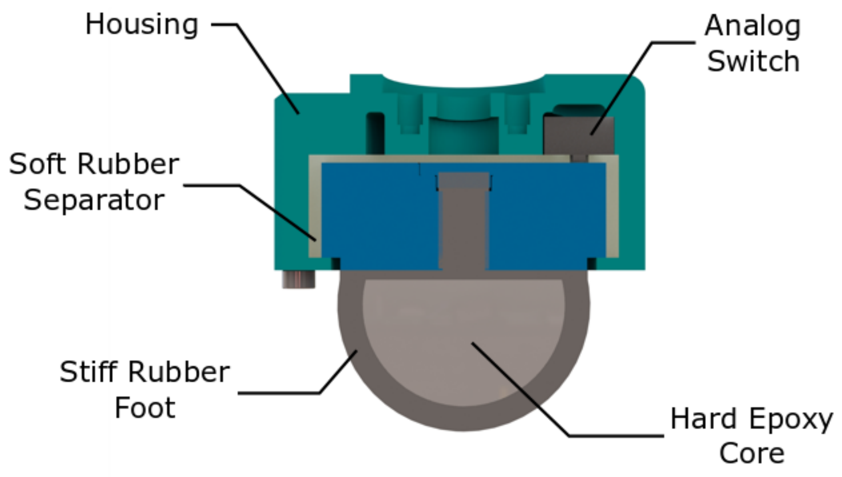


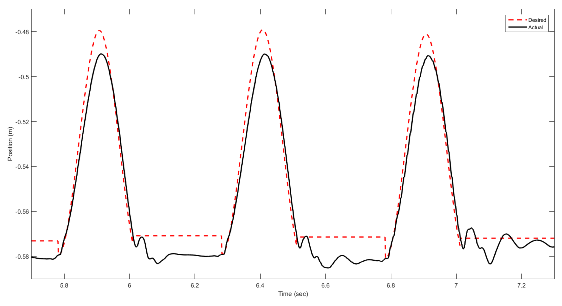
Figure 4.7: A simplified diagram of the foot showing the different components that allow for contact sensing.

In addition to the problems created by contact detection the control of the foot is also an issue due to the flexibility in the belt transmission. The belt transmission, discussed in Chapter 2, uses a fiberglass reinforced timing (toothed) belt that has almost no stretch. Although, the belt loop itself is reinforced with fiberglass the teeth themselves are not reinforced which result in a non-linear backlash effect. During the decent portion of the swing phase the foot will first accelerate towards the ground causing the teeth to stretch in the direction of torque application. Mid-way through the decent portion the controller will decelerate the foot in preparation for ground contact by switching the direction of torque, causing the belt teeth to stretch in the opposite direction. This deceleration and stretching of the teeth will cause the foot to elongate beyond that of the commanded limb length causing the foot to hit the ground prematurely.

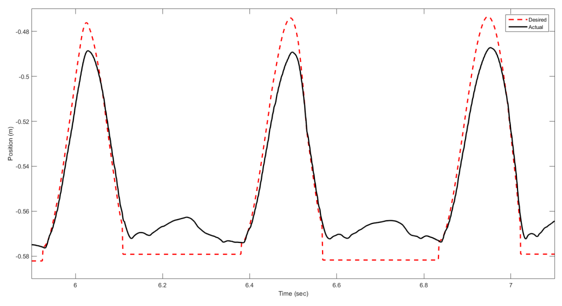
Originally, the controller commanded the foot to hold position at the point of contact. However, this caused instability because on impact the limb is in an elongated pose causing detection of the ground early, meaning that the commanded holding position was actually above the ground once the limb returned to its compressed position under the weight of the robot. The solution to this problem was to command a holding position 1.5 cm into the ground after contact was detected, a comparison of the two methods is shown in Fig. 4.8. This is believed to have two effects, the first being that this position is closer to the actual position that the limb will be in once the belt has stretched under the weight of the robot. Second, this applies a virtual pulling force into the ground helping to force the foot to remain in contact with the ground. Changing the desired holding position upon contact reduced the amount of bounce significantly and increased the stability of the controller.

4.3.6 Discussion and Results

All of the test performed on hardware were done using the hardware described in Chapter 2. The tests performed in simulation were done on a Razer Blade laptop with an Intel i7-7700HQ CPU at 2.80 GHz with 16.0 GB of RAM. All tests were done using Gurobi 8.1 [59] as the QP solver with the same MPC parameters shown in Table 4.1. The MPC used the direct method for the QP formulation and the extended state space that included swing leg dynamics. The simulations were performed in a Gazebo environment using an accurate



(a)



(b)

Figure 4.8: The desired foot trajectory is in the dashed red line and the actual trajectory is in the solid black line. a) is the method in which the foot is commanded to stay at the touchdown position. b) is the method in which the foot is commanded 1.5cm below the touchdown position.

model of the robot from SolidWorks.

m	17.55 kg	Θ weight	1.0
I_{xx}	0.845 kgm^2	x position weight	3.0
I_{yy}	1.296 kgm^2	y position weight	3.0
I_{zz}	0.75 kgm^2	z position weight	50.0
g	9.81 m/s^2	ω weight	0.0
μ	0.4	x velocity weight	0.27
f_{min}	10 N	y velocity weight	0.27
f_{max}	200 N	z velocity weight	0.0
β yaw blending	0.7	α force weight	1×10^{-6}

Table 4.1: Parameters used for the MPC

4.3.6.1 Hardware Testing

The first test that was performed was a lateral push test showing the recovery ability of the MPC. In this test a push in the +Y direction of the robot frame was applied to the robot. The robot was then given time to recover and then a second lateral push in the -Y direction was applied. The results of the test are shown in Fig. 4.9-4.10, please note that the data was recorded in the inertial frame which had a yaw offset from the robot frame which is why the disturbances do not line up with the Y direction. The lateral pushes caused a 0.7 m/s lateral velocity that the controller needed to mitigate. The desired linear positions change when the error between the desired position and actual position exceeds 0.1 m but, all of the other desired trajectories remain at zero. The controller responds in a very intuitive way which is to lean against the push; in the inertial frame the first push was in the +Y and -X directions and the controller responded by commanding a positive roll and pitch. In addition to changing the robot's posture, the robot takes recovery steps once the velocity error is above 0.15 m/s. The footstep are planned using the method discussed in Section 3.1, which uses capture point heuristics to help counter act the applied disturbance. In both cases the robot was able to fully recover through the use of the MPC and recovery footstep planning. The next test that was performed was a walking test using a trotting

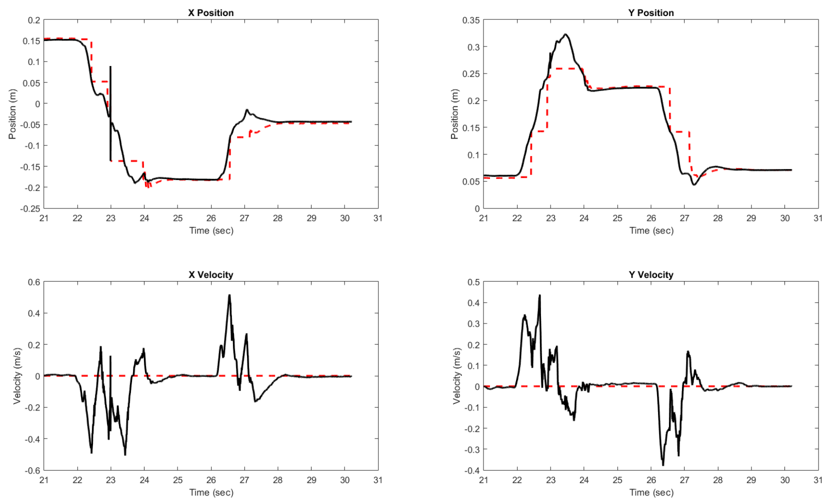


Figure 4.9: Shows the linear positions and velocities of the robot during two lateral pushes 2.5 seconds apart in the opposite direction. The red dashed line is the desired trajectory and the solid black line is the actual trajectory.

gait with 0.25 second swing and stance phases. The results of the test are shown in Fig. 4.11. From these results it is clear that the robot was able to successfully follow the desired trajectory. The test was performed on a padded surface in order to provide the feet with

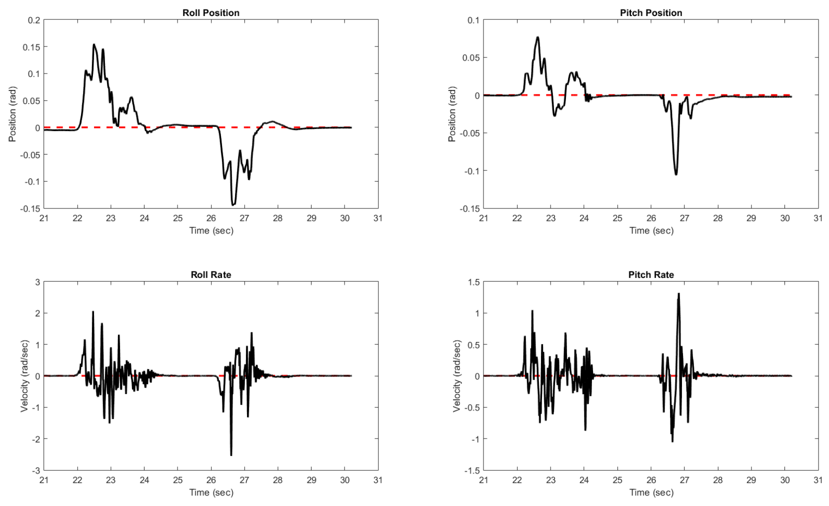


Figure 4.10: Shows the orientation and angular rates of the robot during two lateral pushes 2.5 seconds apart in the opposite direction. The red dashed line is the desired trajectory and the solid black line is the actual trajectory.

more damping. As discussed in the previous section foot bouncing on impact was an issue and a foot redesign is required, however for this testing pads were used to mitigate this issue. Tests at higher velocities were attempted but foot contact detection became an issue. There are two different ways of achieving higher velocities, the first is to increase the stride length and keep the stance time the same. The second, is to decrease the stance and swing time and keep the stride length the same. Due to the kinematic configuration of ALPHRED the ability to increase stride length is severely limited. With a traditional quadruped the leg is capable of going under the robot without complications, as ALPHRED's foot gets closer to the body the hip yaw has to make large movements to produce small changes in the foot position. For this reason ALPHRED's strides do not go under the body which limits the length. As a consequence decreasing the stance and swing times are the most effective way

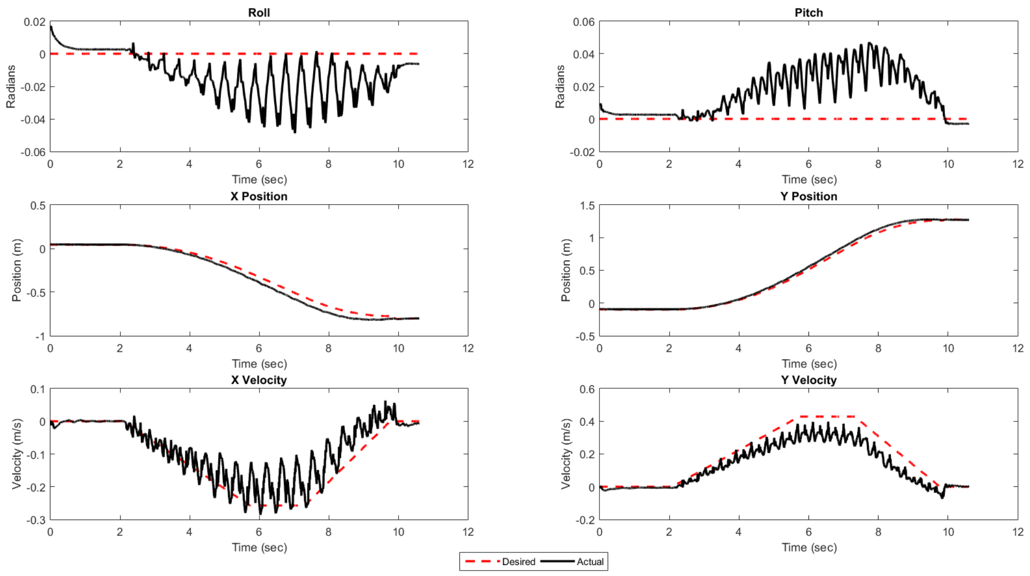


Figure 4.11: These plots show the data gathered from a 0.4 m/s trot.

of achieving higher velocities for the ALPHRED platform. Unfortunately, decreasing the swing times requires much higher acceleration profiles causing the foot contact sensors to trigger prematurely. In addition, a larger acceleration profile results in larger touchdown velocities which increases the likelihood of a foot bouncing on impact. Due to these factors it was difficult perform trotting with swing times lower than 0.25 seconds.

4.3.6.2 Simulation Results

In simulation the robot was able to perform a flying trot with 0.12 seconds stance time and a 0.16 seconds swing time. The flying trot was able to achieve a maximum velocity of 1 m/s as shown in Fig. 4.12. The robot was able to track the desired trajectory quite well even the extremely large deceleration at the end. The MPC naturally pitched the body forward slightly in the direction of motion. When velocities of greater than 1 m/s were commanded the robot would reach kinematic limits. From the position control PD in Section 4.2 we know that the robot is capable of speeds of +1.5 m/s, however with this controller it was easier to directly control the position of the body relative to the feet. I believe that by adding a kinematic constraint larger velocities are achievable.

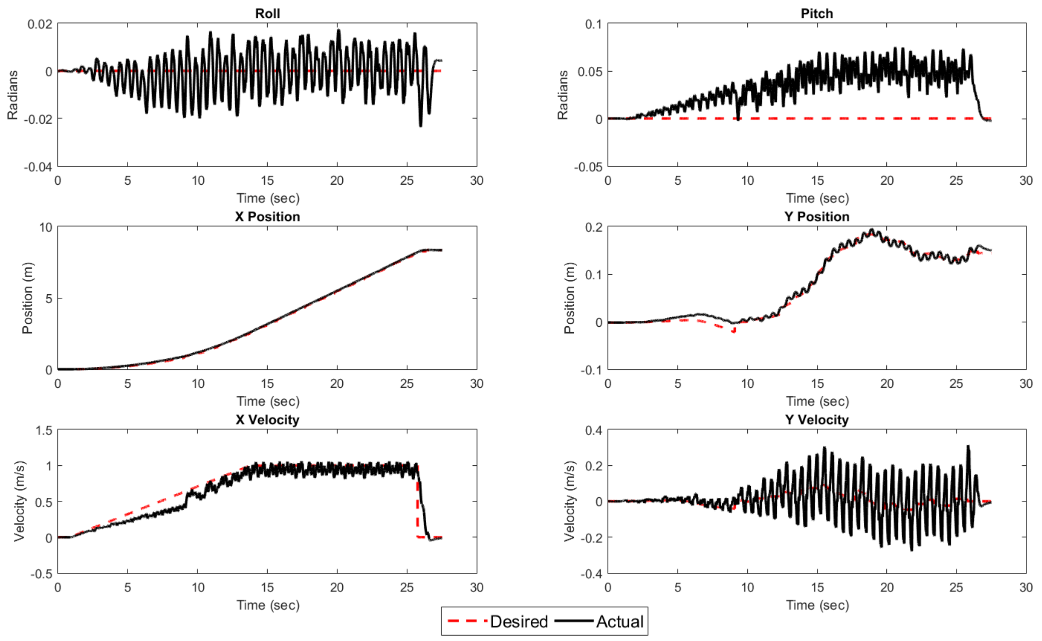


Figure 4.12: Data from a flying trot in simulation. The stance time is 0.12 seconds and the swing time is 0.16 seconds.

4.3.6.3 Contribution of Swing Leg Dynamics

ALPHRED is built with low leg inertia to body inertia and for this reason the contributions made by the swing leg are relatively small. To highlight the contributions made by including the swing leg dynamics a simulation was ran with heavy feet by increasing the mass of the tibia from 0.495 Kg to 2.02 Kg and their inertia tensor accordingly. Two gaits were tested using the heavy feet, an amble gait (one foot off the ground at a time) Fig. 4.13 and a trot gait Fig. 4.14. For both gaits the major effect that including the swing leg dynamics had was it helped to stabilize the yaw position. Considering that the yaw axis was difficult to stabilize due to they dynamics of the robot it is not surprising to see that the yaw position was effected the most by the heavy feet. In the amble gait the controller that did not account for the swing leg dynamics actually went unstable whereas the one that did account for the swing leg dynamics did not. In both cases there was also a reduction in the variance of the Z height of the robot. It should be noted that the computation of the swing leg dynamics are non-negligible and add approximately 1 ms to the formulation time. The direct formulation without the swing leg dynamics runs at about 250-350 Hz whereas if the swing leg dynamics are accounted for the frequency drops to 100-250 Hz.

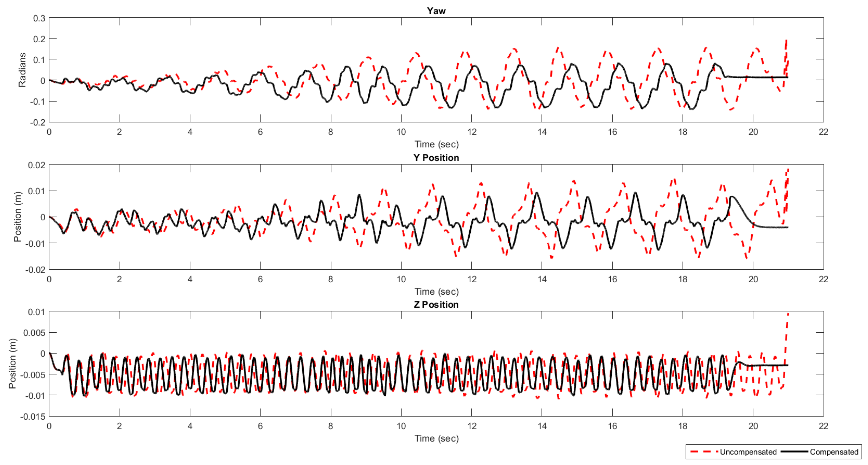


Figure 4.13: Shows the error between the desired trajectory and the actual trajectory for an amble gait with heavy feet in simulation. MPC that doesn't include swing leg dynamics is represented by the dashed red line and MPC that does include swing leg dynamics is in the solid black line.

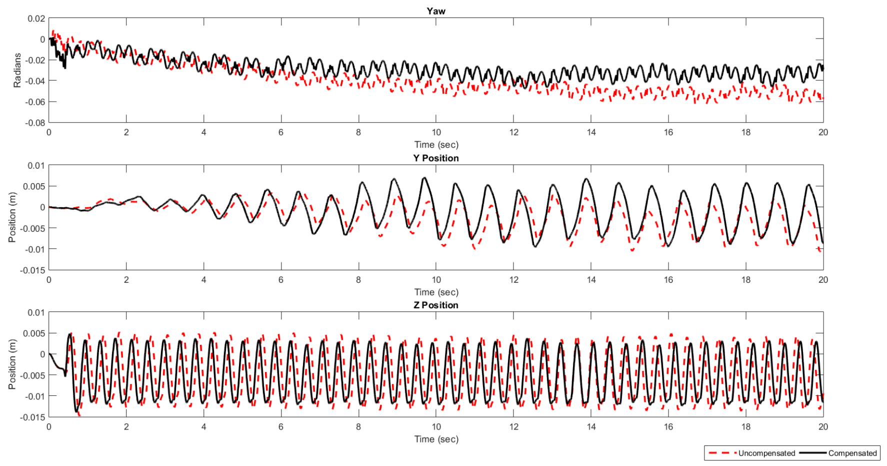


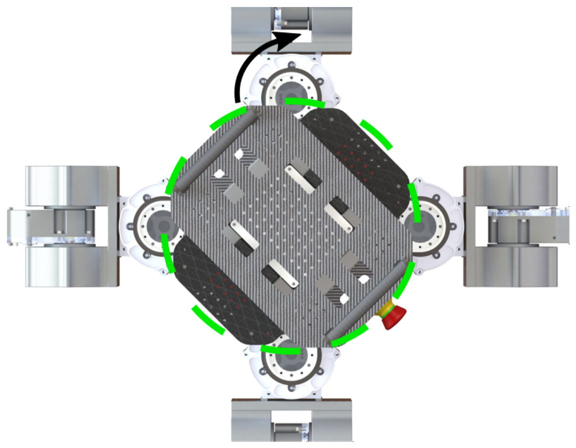
Figure 4.14: Shows the error between the desired trajectory and the actual trajectory for a trot gait with heavy feet in simulation. MPC that doesn't include swing leg dynamics is represented by the dashed red line and MPC that does include swing leg dynamics is in the solid black line.

The results from simulation are encouraging and lead me to believe that this method will work one robots that are not optimized with low inertia legs and even be used to extend this type of controller to humanoid robots. In addition, I believe accounting for swing leg dynamics will also allow for preplanned dynamic tasks such as a large leap or kicking a ball. The act of kicking a ball requires a humanoid robot to balance on one leg while swinging the kicking leg at a high angular rate. If the original controller is used then the large dynamics created by the swing leg is ignored whereas if the kicking motion is preplanned then the entire kicking motion could possibly be stabilized using the MPC developed in this work.

CHAPTER 5

MANIPULATION

The previous chapters described the control required to achieve locomotion comparable to the most advanced quadrupeds in the world, even with ALPHRED's non-conventional kinematic configuration. However, what makes ALPHRED truly unique is the robots ability to use it's limbs as manipulators. Currently there are multiple quadrupedal platforms in the world that are capable of dynamic and robust locomotion, getting closer and closer to being able to traverse any type of terrain. However, none of these platforms are capable of any other task besides surveillance and inspection. With ALPHRED we have preserved the locomotion capabilities of a traditional quadruped but we have also provide ALPHRED the capability of picking up packages by using it's limbs as manipulators. This capability is due to ALPHRED's kinematic configuration and the use of proprioceptive actuators. As discussed earlier ALPHRED's kinematic chain has two major differences from the traditional quadrupedal design: The first is the change from a hip roll degree of freedom to a hip yaw degree of freedom, and the second being the radially symmetric layout of the body, Fig. 5.1. These two changes allow opposite limbs to become parallel, providing the capability of duel limb manipulation. Accurate force control is required for two limbs to successfully manipulate the same object simultaneously, which is why the proprioceptive actuators are


(a)

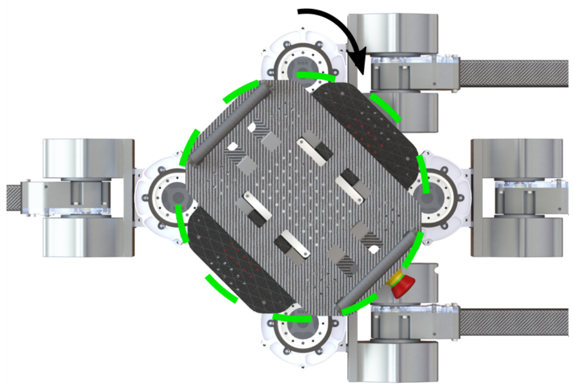

(b)

Figure 5.1: The radially symmetry of the limb layout combined with the hip yaw degree of freedom provides the range of motion to allow two opposing limbs to become parallel as shown in b).

essential for ALPHRED's manipulation capabilities. The subsequent sections describe the methods used to allow for package pick-up and drop-off capabilities.

5.1 End-Effector Design

Each limb's end-effector has an additional actuated degree of freedom that different tools can be added to. For the limbs that are used in the stance phase of the bi-pedal mode the attachments consist of aluminum wings. When the wings are in the up position the limb is a point contact that can be used for dynamic locomotion. However, when the wings are in the down position the foot acts like a 24.5 cm bar increasing the support polygon so that the robot can easily balance on two feet.

There are three unique attachments that are added to the limbs that are used for manipulation shown in Fig. 5.2. The first is a circular pad connected to a passive rotational joint. This tool is used to clamp onto the flat surface of a box. The passive joint is aligned perpendicular to the actuated degree of freedom so that the torque from the actuated joint acts on the box but the passive joint will keep the pad parallel to the box surface as the box is transferred to the back of the robot. This allows the robot to control the pitch of the box keeping it upright for the entire process. The attachment opposite the circular pad is a smooth hook like tool. This tool can be inserted into the loop of a handle on an object like a bag. Once inserted the hook is rotated upwards so that the handle is not resting in the pocket that is made between the tool and the rest of the end-effector. As the bag is transferred to the back of the robot the hook will be commanded to remain vertical keep the handle of the bag securely in the pocket. Once the bag is on the bag the tool will be rotated downward so that the handle will naturally slip off of the smooth surface. The final tool added to the end-effector is a Sharp GP2Y0A21YK infrared proximity sensor. This sensor is used to provide the robot with distance information so that the robot can determine where the package is and how large it is.

5.2 Hybrid Force-Position Control

This section details the controllers and methods developed to enable ALPHRED 2 to be able to handle packages of different sizes, weights, and materials. All controllers are done in the package coordinate frame which is depicted in Fig. 5.3.

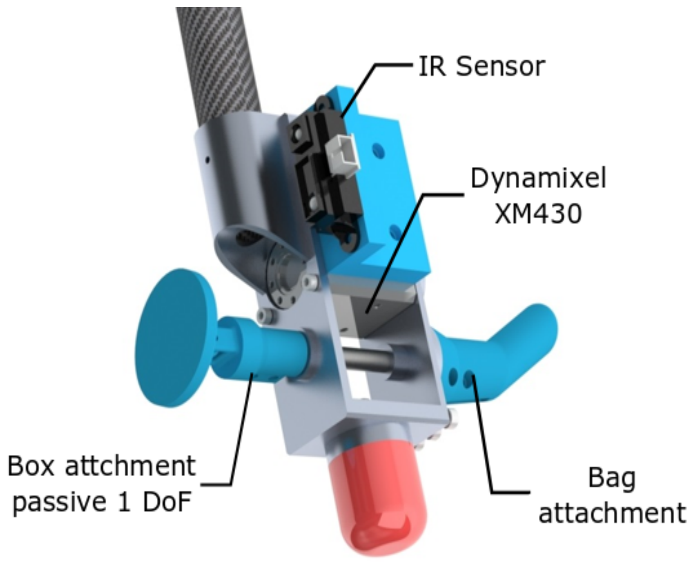


Figure 5.2: Rendering of the end-effector with all manipulation attachments.

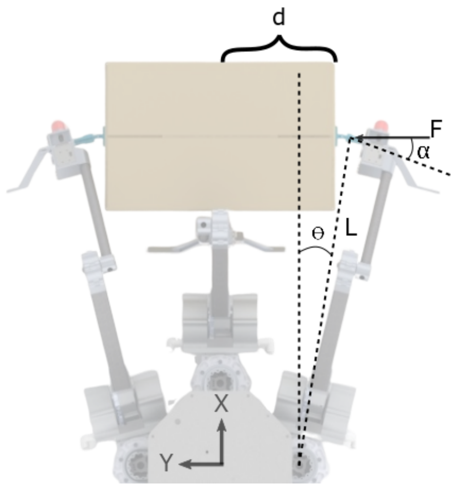


Figure 5.3: Diagram of the package frame used to transfer the package from the ground to the top of the robot.

5.2.1 Auto Clamping

Boxes not only come in different shapes and sizes but also different stiffnesses. For this reason, an auto clamping routine was developed to appropriately calculate the required clamping force for a given box. First, ALPHRED 2 is commanded to close the gap between its two manipulators with a max yaw torque of 3 Nm. The robot detects a successful clamping once a significant position error is detected. At this point the max yaw torque is slowly increased until the position error begins to decrease. A decreasing position error means that the package is beginning to flex and the appropriate clamping force can be calculated from (5.1).

$$F = \frac{cos(\alpha)\tau}{L} \tag{5.1}$$

Where F is the clamping force, L is the horizontal distance from the yaw actuator to the manipulator, τ is the torque from the yaw actuator, and α is the angle between the force created by the moment arm and the normal vector of the box surface as shown in Fig. 5.3

5.2.2 Pseudo Force-Position Control

A constant clamping force is crucial to the success of ALPHRED 2 transferring the package from the ground or table to its back. This requires that the X and Z position be controlled while simultaneously controlling the Y force, Fig. 5.4. For this reason, a pseudo hybrid force and position controller was developed similar to those discussed in [68]. Once the manipulators are clamped onto the box the Y distance d from the yaw actuator to the manipulator and the Z distance Z_0 from the ground to the end effectors are calculated. A X and Z trajectory, in the package frame, is then determined that will move the package from the ground to a height of Z_0 above the top of the robot. Along the entire trajectory the Y value is commanded to be $d - 0.01$ which will cause the end-effector to move 1 cm into the package. At every time step the max torque of the actuator is be calculated and set via (5.1). Since the manipulator is always being commanded into the box the actuator's low level controller will register a large position error ensuring that the maximum torque is applied resulting in the appropriate clamping force.

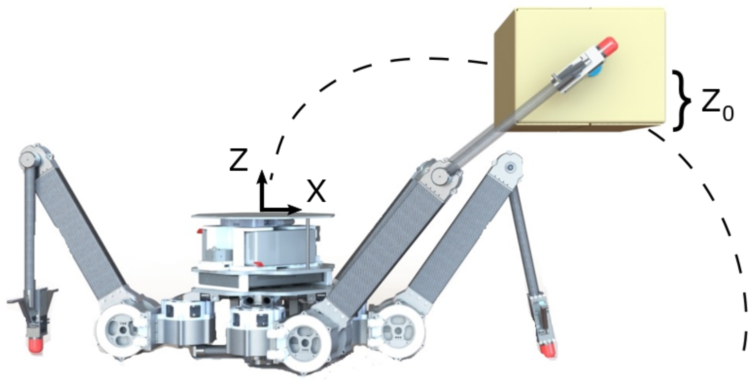


Figure 5.4: The position is controlled in the X and Z plane while the force is maintained in the Y direction to ensure that the package does not slip during transfer. The distance z_0 from the ground to the pickup point is used to determine the distance from the top of the robot to drop off the package.

Most of the current hybrid control schemes are done in torque mode whereas the one just described is done in position mode. The main reason for choosing position control over force control is for safety reasons. If ALPHRED attempts to pick up a package that is too heavy or the surface is slippery than a slip will occur and the package will fall out of the robot's hands. With the position controlled scheme the end-effectors will move 1 cm inwards and the robot will know the package slipped. With a force control scheme if the slip is not detected the end-effectors will slam together potentially causing damage to the robot.

5.2.3 Auto-Scan

In order to pick up a package successfully the end-effectors must be close to the packages CoM. One way of achieving this is via tele-operation by the user, however ALPHRED can also automatically determine where the center of the box is by scanning the package using the IR sensors equipped at the end-effectors. First, the end-effectors are moved close to the package to ensure the IR sensors have consistent readings (5 cm from package). From these measurements, the robot can approximate the width of the package. Next, the end-effectors are moved lineally in the +Z direction until the package is no longer detected. From this point, the package height can be approximated using the forward kinematics of the end-effectors. Finally, the X direction is scanned and the length of the package is determined. From these approximations the CoM is assumed to be in the geometric center of the package and ALPHRED will position its end-effectors accordingly.

There are many times that the length or height of the package is out of reach of ALPHRED however the package is perfectly adequate for the robot to transport. In this case if ALPHRED knows a-priori information about the package then from the two sides that it could successfully scan the robot can deduce the dimension of the third side and determine the correct geometric center of the box.

5.2.4 Online Learning of Parameters

As ALPHRED is going through the automated routine of picking up a package the robot can also estimate information about the package as it goes. In the auto scanning process ALPHRED determines the dimensions of the package. From the auto-clamping routine the robot ascertains the stiffness of the package. In addition, the robot also approximates the mass of the package. This is done by having the robot lift the package off of the ground and pausing for a moment, from this static position the mass of the package can be estimated from the geometry of the pose and the commanded torque at the hip pitch actuator. Assuming that the package has uniform density we can approximate the inertia tensor of the box using (5.2), which can be passed to the locomotion controller in Chapter 4.

$$I_{box} = \begin{bmatrix} \frac{1}{12}m(y^2+z^2) & 0.0 & 0.0 \\ 0.0 & \frac{1}{12}m(x^2+z^2) & 0.0 \\ 0.0 & 0.0 & \frac{1}{12}m(x^2+y^2) \end{bmatrix} \tag{5.2}$$

5.3 Full Procedure

The full procedure of autonomously picking up a parcel is as follows:

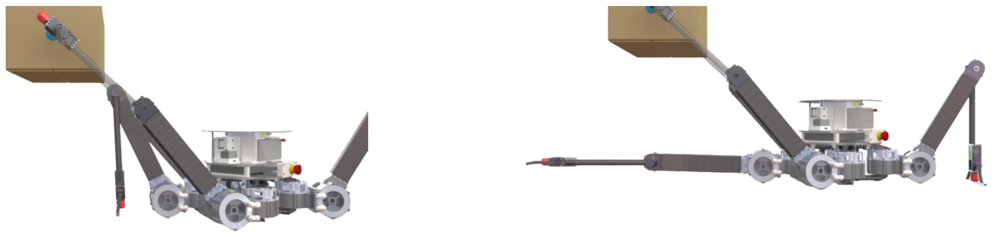
1. ALPHRED positions itself relative to the box with one directly aimed at the box.
2. If the box is on the ground ALPHRED will lower himself onto his belly or if the box is off of the ground ALPHRED will transition into bipedal mode balancing on the limb directly infront of the box and the opposite limb.
3. The two adjacent limbs to the one in front of the box will perform the auto-scanning

routine (Section 5.2.3).

4. Those same two limbs will then grasp the box at the box's estimated CoM using the auto-clamping routine to securely grasp the box and determine it's stiffness (Section 5.2.1).

5. If the robot is on the ground then interference with the front leg is of concern. For this reason, the robot will lift the parcel, using the pseudo force-position control (Section 5.2.2), away from the body to allow the front leg to straighten out allowing the parcel to be placed onto the robots back without interference (Fig. 5.5). If the robot is in biped mode the front leg interference is not of concern.

6. ALPHRED then walks to the desired location and repeats those actions in reverse while using the parameters estimated from the first lift to place the box in a new location.



(a) Front leg tucked pose.

(b) Front leg stretched out pose.

Figure 5.5: Transition from the leg tucked pose to the leg stretched out pose to ensure no interference when the robot is placing the package on it's back.

5.4 Discussion and Results

To test ALPHRED's manipulation capabilities the robot used the procedure described in the previous section to attempt to pick up boxes of varying size. Table 5.1 is a list of all boxes that ALPHRED was able to successfully pick up. The smallest box by volume was box 9 at $2,323cm^3$ and the largest was box 1 at $28,980cm^3$. If the limbs go past 90°, meaning that d is smaller than the distance from the CoM to the hip yaw axis is Fig. 5.3, when the package is grasped then the robot will be unable to transfer the package to the top of the robot. This is due to the geometry of the package pickup problem, which when the angle is greater than 90° will cause the yaw angle to grow more in order to keep contact with the box during the transition. Whereas, if the angle starts below 90° then the angle will shrink during transportation. For this reason the smallest width that ALPHRED can grasp is 10 cm. The largest length and height dimensions are dependent on each other and are determined by the front leg interference issue shown in Fig. 5.5. During the experiment box 1 grazed the top of the robot's knee thus the largest diagonal created by the length and height plane is 19.3 cm.

In addition to size testing weight testing was also performed. The 4th box down the list of Table 5.1, 31 cm x 21.5 cm x 14 cm, was considered to be the nominal box size. The weight in this box was increased until the box slipped out of ALPHRED's grip. ALPHRED successfully lifted boxes ranging from completely empty all the way up to 3 Kg. All of these tests were performed with ALPHRED on the ground. In bipedal mode ALPHRED was able to successfully pickup/dropoff box 4 with a 1.5 Kg payload from a height of 1.02 m down to a hieght of 0.0145 m. All failures were due to insufficient grasping force rather than the

Table 5.1: Box Sizes Successfully Lifted

No.	Size (cm^3)
1.	48 x 34.5 x 17.5
2.	41.5 x 33 x 14
3.	39 x 32 x 27
4.	31 x 21.5 x 14
5.	26 x 19.5 x 14
6.	25.4 x 17.8 x 13.3
7.	23 x 23 x 10
8.	22 x 21 x 13.5
9.	16 x 13.2 x 11

robot not being able to support the loads.

CHAPTER 6

FUTURE WORKS AND CONCLUSIONS

6.1 Future Work

The advancements made in this work has lead to many areas that future work could be made to further advance the capabilities of legged robotics. A number of potential areas are described below.

- As noted in Section 4.3.5 the stretch in the belt caused a number of non-linearities that made the transition from the swing phase to the stance phase difficult to control. Additionally, any flexibility in the system causes inaccuracies in both the position and torque control of the limb which results in decreased control authority of the entire robot. I believe that the leg should be redesigned with a linkage system rather than a belt or chains transmission. ALPHRED requires $+180°$ of rotation at the knee, therefore a traditional linkage system will not be adequate. However, if two four-bar linkages are used at the knee forming an antagonistic pair then the desired range of motion can be achieved.

- Currently ALPHRED is only able to pickup packages that don't require the hip yaw actuators to go past $pm90^{circ}$ which limits the minimum width of the package to be 10 cm. To overcome this limitation the manipulation end-effector should be redesigned to incorporate a small pitch lead screw attached to the gripping pads. With this design the distance of the gripping pad could be extend to grip packages smaller than 10 cm in width. However, by using a small pitch the orientation of the package could still be controlled without disengaging from the package.

- During the standard trot gait ALPHRED's hip pitch actuator uses 70% of the actuators peak torque. The reason for this is due to the parallel configuration of ALPHRED's limbs. In the parallel configuration the hip pitch actuator needs to provide the opposite torque of the knee actuator plus the torque required at the hip, resulting in the hip pitch working much harder than all other actuators. In nature most legged creatures use 20%-30% over there total actuation torque during nominal locomotion. The additional torque is used to recover from large unexpected disturbances. To correct for this issue the hip actuator should be upgraded to an actuator with double the peack torque. This can easily be done by upgrading the motor in the BEAR actuator from a U8 to a U10.

- This work focused on the difficult task of developing the techniques to successfully pick up and drop off packages. Magnets inside of the package and on the top of the robot were used to secure the package during the transportation period. However, to implement end-to-end package delivery in the real world a method to secure the package to the top of the robot must be developed. I believe that a lightweight single degree of freedom grasper can be developed to clamp onto the package during transportation. In addition, this method could be used to position the package closer to the CoM to improve the dynamics of the robot while transporting the package.

- One of the biggest contributions to this work was the reformulation of the MPC from an indirect method to a direct method to achieve cycle times of +250 Hz. However, the current formulation still have inefficiencies that could be improved. If a foot is in swing phase the upper and lower bounds are commanded to be zero telling the QP that this foot is not allowed to support weight. However, if the foot is not supporting weight it can be removed from the optimization problem completely potentially reducing the number of decision variables. This method would however decrease the performance of the warm start because the size of the problem would change from time step to time step. Additionally, the current implementation used Gurobi to solve the QP problem. Gurobi is one of the best QP solvers in the world and does extremely well at solving some of the most difficult benchmarks. However, the QP problem being solved is relatively easy to solve and therefore a QP solver with less overhead could be used to reduce some of the time to solve the problem.

- This work expanded on trajectory planning algorithms that used ZMP dynamics to solve for viable CoM trajectories and footstep locations. Whereas, the MPC used a linearized floating base model to solve for desired ground reaction forces. I believe that the trajectory optimization algorithms could use a non-linear floating base model that better compliments the MPC to solve for CoM trajectories, footstep locations, and ground reaction forces that could then be passed to the MPC. A similar TO algorithm is discussed in [88].

- One of the biggest difficulties for highly dynamic locomotion was accurately detecting contact with the ground. Due to the mechanical design of the foot contact switch false positives would be detected during high acceleration foot swings. One way to improve this is by reducing the mass of the foot contact sensor which will result in larger accelerations need for the same actuation force. Additionally contact estimators using a Kalman filter [13] or machine learning [63] could also be added to improve the accuracy of touchdown detection.

- Currently the incorporating the swing leg dynamics was only found to be effective in simulation. In the future I would like to verify this by adding weight to the feet of ALPHRED or performing very dynamic motions that would otherwise be unstable without compensating for the swing leg dynamics.

6.2 Conclusions

Robots have benefited mankind in countless ways from the manufacturing floor to assembly lines. As technology progresses roboticists and industry leaders have been pushing for robots to step out from the factory floors and into the human world. One of the most immediate uses is delivery systems to solve the last mile problem. The United States Postal Service (USPS) shipping services covered over 6.2 billion packages in 2018 [78]. For many of these parcels the last mile of the delivery process is a large portion of the cost reaching and even exceeding 50% of the total cost, making it a crucial component to delivery[41]. This work presented a quadrupedal robot that used a novel kinematic design and an innovative control strategy to show that legged robotics can be part of the solution to this problem.

ALPHRED is one of the first robots that uses a multi-modal approach to handle a variety of tasks. Using it's quadrupedal locomotion mode and a simple position control framework, ALPHRED is able to traverse uneven terrain and trot at 1.5 m/s. The developed MPC showed exciting robustness when the foot contact sensing was working as expected. In addition, extending the MPC to account for swing leg dynamics showed promising results in simulations, and leads me to believe that the developed MPC could be extended to the control of humanoid robots. Beyond quadrupedal locomotion ALPHRED is capable of using it's unique kinematic configuration to operate in bipedal mode to pickup a package. Looking ahead to real world situations where the robot may need to traverse long flat terrains such as sidewalks or hallways, ALPHRED can use its caster mode to push itself along using a highly energy efficient form of locomotion. Finally, ALPHRED can transform back into it's bipedal mode to successfully realize the task of end-to-end package delivery. I believe that this work shows the potential for legged robotics to someday step out of the factories and into the real world.

Bibliography

[1] BostonDynamics. https://www.bostondynamics.com/. Accessed: 2019-11-24.

[2] Evan Ackerman. Agility robotics introduces cassie, a dynamic and talented robot delivery ostrich, 2017.

[3] Aaron D Ames. Human-inspired control of bipedal walking robots. *IEEE Transactions on Automatic Control*, 59(5):1115–1130, 2014.

[4] Taylor Apgar, Patrick Clary, Kevin Green, Alan Fern, and Jonathan W Hurst. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems*, 2018.

[5] C Dario Bellicoso, Christian Gehring, Jemin Hwangbo, Péter Fankhauser, and Marco Hutter. Perception-less terrain adaptation through whole body control and hierarchical optimization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 558–564. IEEE, 2016.

[6] C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. Dynamic locomotion and whole-body control for quadrupedal robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3359–3365. IEEE, 2017.

[7] C Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots. *IEEE Robotics and Automation Letters*, 3(3):2261–2268, 2018.

[8] John T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March 1998.

[9] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam, 2010.

[10] Pranav A Bhounsule, Jason Cortell, and Andy Ruina. Design and control of ranger: an energy-efficient, dynamic walking robot. In *Adaptive Mobile Robotics*, pages 441–448. World Scientific, 2012.

[11] Daniel J Blackman, John V Nicholson, Camilo Ordonez, Bruce D Miller, and Jonathan E Clark. Gait development on minitaur, a direct drive quadrupedal robot. In *Unmanned Systems Technology XVIII*, volume 9837, page 98370I. International Society for Optics and Photonics, 2016.

[12] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

[13] Gerardo Bledt, Patrick M Wensing, Sam Ingersoll, and Sangbae Kim. Contact model fusion for event-based locomotion in unstructured terrains. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[14] Michael Bloesch, Christian Gehring, Péter Fankhauser, Marco Hutter, Mark A Hoepflinger, and Roland Siegwart. State estimation for legged robots on unstable and

slippery terrain. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6058–6064. IEEE, 2013.

[15] Michael Bloesch, Marco Hutter, Mark A Hoepflinger, Stefan Leutenegger, Christian Gehring, C David Remy, and Roland Siegwart. State estimation for legged robots-consistent fusion of leg kinematics and imu. *Robotics*, 17:17–24, 2013.

[16] Francesco Bullo and Richard M Murray. Proportional derivative (pd) control on the euclidean group. In *European Control Conference*, volume 2, pages 1091–1097, 1995.

[17] John J Craig. *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[18] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 279–286. IEEE, 2014.

[19] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[20] Dimitar Dimitrov, Alexander Sherikov, and Pierre-Brice Wieber. A sparse model predictive control formulation for walking motion generation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2292–2299. IEEE, 2011.

[21] M. Freese E. Rohmer, S. P. N. Singh. V-rep: a versatile and scalable robot simulation

framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[22] Johannes Englsberger, Christian Ott, Máximo A Roa, Alin Albu-Schäffer, and Gerhard Hirzinger. Bipedal walking control based on capture point dynamics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4420–4427. IEEE, 2011.

[23] Roy Featherstone. A beginner's guide to 6-d vectors (part 1). *IEEE robotics & automation magazine*, 17(3):83–94, 2010.

[24] Roy Featherstone. A beginner's guide to 6-d vectors (part 2)[tutorial]. *IEEE robotics & automation magazine*, 17(4):88–99, 2010.

[25] Roy Featherstone. *Rigid body dynamics algorithms.* Springer, 2014.

[26] Siyuan Feng. Online hierarchical optimization for humanoid control. 2016.

[27] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[28] Michele Focchi, Andrea Del Prete, Ioannis Havoutis, Roy Featherstone, Darwin G Caldwell, and Claudio Semini. High-slope terrain locomotion for torque-controlled quadruped robots. *Autonomous Robots*, 41(1):259–272, 2017.

[29] Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. Compliant leg behaviour

explains basic dynamics of walking and running. *Proceedings of the Royal Society B: Biological Sciences*, 273(1603):2861–2867, 2006.

[30] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[31] Michael Grant, Stephen Boyd, and Yinyu Ye. Cvx: Matlab software for disciplined convex programming, 2008.

[32] Robert J Griffin, Georg Wiedebach, Sylvain Bertrand, Alexander Leonessa, and Jerry Pratt. Straight-leg walking through underconstrained whole-body control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE, 2018.

[33] Jessy W Grizzle, Christine Chevallereau, Aaron D Ames, and Ryan W Sinnet. 3d bipedal robotic walking: models, feedback control, and open problems. *IFAC Proceedings Volumes*, 43(14):505–532, 2010.

[34] Jessy W Grizzle, Jonathan Hurst, Benjamin Morris, Hae-Won Park, and Koushil Sreenath. Mabel, a new robotic bipedal walker and runner. In *2009 American Control Conference*, pages 2030–2036. IEEE, 2009.

[35] Gael Guennebaud, Benoit Jacob, et al. Eigen: a c++ linear algebra library. *URL http://eigen. tuxfamily. org, Accessed*, 22, 2014.

[36] Joshua Hooks and Dennis Hong. Implementation of a versatile 3d zmp trajectory optimization algorithm on a multi-modal legged robotic platform. In *2018 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 3777–3782. IEEE, 2018.

[37] Donald F Hoyt and C Richard Taylor. Gait and the energetics of locomotion in horses. *Nature*, 292(5820):239–240, 1981.

[38] Ian W Hunter, John M Hollerbach, and John Ballantyne. A comparative analysis of actuator technologies for robotics. *Robotics Review*, 2:299–342, 1991.

[39] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44. IEEE, 2016.

[40] Juan L Jerez, Eric C Kerrigan, and George A Constantinides. A condensed and sparse qp formulation for predictive control. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 5217–5222. IEEE, 2011.

[41] Martin Joerss, Jürgen Schröder, Florian Neuhaus, Christoph Klink, and Florian Mann. Parcel delivery the future of last mile. *McKinsey & Company*, 2016.

[42] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to humanoid robotics*, volume 101. Springer, 2014.

[43] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada,

Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *ICRA*, volume 3, pages 1620–1626, 2003.

[44] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. 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)*, volume 1, pages 239–246. IEEE, 2001.

[45] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Fast, robust quadruped locomotion over challenging terrain. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2665–2670. IEEE, 2010.

[46] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Learning, planning, and control for quadruped locomotion over challenging terrain. *The International Journal of Robotics Research*, 30(2):236–258, 2011.

[47] Takeo Kanade, Pradeep K Khosia, and Nobuhiko Tanaka. Real-time control of cmu direct-drive arm ii using customized inverse dynamics. In *The 23rd IEEE Conference on Decision and Control*, pages 1345–1352. IEEE, 1984.

[48] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301. IEEE, 2019.

[49] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[50] Gavin Kenneally, Avik De, and Daniel E Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2):900–907, 2016.

[51] Donghyun Kim, Ye Zhao, Gray Thomas, Benito R Fernandez, and Luis Sentis. Stabilizing series-elastic point-foot bipeds using whole-body operational space control. *IEEE Transactions on Robotics*, 32(6):1362–1379, 2016.

[52] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, Emanuel Todorov, Olivier Stasse, Maren Bennewitz, and Nicolas Mansard. Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3346–3351. IEEE, 2015.

[53] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[54] Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2589–2594. IEEE, 2014.

[55] Derek Lahr, Viktor Orekhov, Bryce Lee, and Dennis Hong. Early developments of a parallelly actuated humanoid, saffir. In *ASME 2013 international design engineering technical conferences and computers and information in engineering conference*, pages V06BT07A054–V06BT07A054. American Society of Mechanical Engineers, 2013.

[56] Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[57] Michael Neunert, Farbod Farshidian, Alexander W Winkler, and Jonas Buchli. Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters*, 2(3):1502–1509, 2017.

[58] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, 2018.

[59] Gurobi Optimization. Gurobi optimizer 8.0. *Gurobi: http://www. gurobi. com*, 2019.

[60] Matthew P. Kelly. Transcription methods for trajectory optimization: a beginners tutorial. 07 2017.

[61] Diego Pardo, Lukas Möller, Michael Neunert, Alexander W Winkler, and Jonas Buchli. Evaluating direct transcription and nonlinear optimization methods for robot motion planning. *IEEE Robotics and Automation Letters*, 1(2):946–953, 2016.

[62] Hae-Won Park, Patrick M Wensing, and Sangbae Kim. High-speed bounding with the mit cheetah 2: Control design and experiments. *The International Journal of Robotics Research*, 36(2):167–192, 2017.

[63] Stylianos Piperakis, Stavros Timotheatos, and Panos Trahanias. Unsupervised gait

phase estimation for humanoid robot walking. In *IEEE Intl. Conf. on Robotics and Automation*, 2019.

[64] Gill A Pratt and Matthew M Williamson. Series elastic actuators. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 399–406. IEEE, 1995.

[65] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami. Capture point: A step toward humanoid push recovery. In *2006 6th IEEE-RAS international conference on humanoid robots*, pages 200–207. IEEE, 2006.

[66] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

[67] Marc H Raibert. *Legged robots that balance.* MIT press, 1986.

[68] Marc H Raibert and John J Craig. Hybrid position/force control of manipulators. *Journal of dynamic systems, measurement, and control*, 103(2):126–133, 1981.

[69] Alireza Ramezani, Jonathan W Hurst, Kaveh Akbari Hamed, and Jessy W Grizzle. Performance analysis and feedback control of atrias, a three-dimensional bipedal robot. *Journal of Dynamic Systems, Measurement, and Control*, 136(2), 2014.

[70] Luis Sentis and Oussama Khatib. A whole-body control framework for humanoids operating in human environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2641–2648. IEEE, 2006.

[71] Sangok Seok, Albert Wang, David Otten, and Sangbae Kim. Actuator design for high force proprioceptive control in fast legged locomotion. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1970–1975. IEEE, 2012.

[72] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[73] Mark W Spong. Underactuated mechanical systems. In *Control problems in robotics and automation*, pages 135–150. Springer, 1998.

[74] Mark W Spong and Romeo Ortega. On adaptive inverse dynamics control of rigid robots. *IEEE Transactions on Automatic Control*, 35(1):92–95, 1990.

[75] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. In *2018 UKACC 12th International Conference on Control (CONTROL)*, pages 339–339. IEEE, 2018.

[76] Tomohito Takubo, Yoshinori Imada, Kenichi Ohara, Yasushi Mae, and Tatsuo Arai. Rough terrain walking for bipedal robot by using zmp criteria map. In *2009 IEEE International Conference on Robotics and Automation*, pages 788–793. IEEE, 2009.

[77] Nikolas Trawny and Stergios I Roumeliotis. Indirect kalman filter for 3d attitude estimation. *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, 2:2005, 2005.

[78] USPS. 2019 postal facts companion, 2019.

[79] Oskar Von Stryk and Roland Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of operations research*, 37(1):357–373, 1992.

[80] Miomir Vukobratović and J Stepanenko. On the stability of anthropomorphic systems. *Mathematical biosciences*, 15(1-2):1–37, 1972.

[81] Andreas Wächter and L Biegler. Ipopt-an interior point optimizer, 2009.

[82] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IFAC Proceedings Volumes*, 41(2):6974–6979, 2008.

[83] P. M. Wensing and D. E. Orin. High-speed humanoid running through control with a 3d-slip model. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5134–5140, Nov 2013.

[84] Patrick M Wensing and David E Orin. Generation of dynamic humanoid behaviors through task-space control with conic optimization. In *2013 IEEE International Conference on Robotics and Automation*, pages 3103–3109. IEEE, 2013.

[85] Patrick M Wensing, Albert Wang, Sangok Seok, David Otten, Jeffrey Lang, and Sangbae Kim. Proprioceptive actuator design in the mit cheetah: Impact mitigation and high-bandwidth physical interaction for dynamic legged robots. *IEEE Transactions on Robotics*, 33(3):509–522, 2017.

[86] A. W. Winkler, F. Farshidian, M. Neunert, D. Pardo, and J. Buchli. Online walking motion and foothold optimization for quadruped locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5308–5313, May 2017.

[87] A. W. Winkler, F. Farshidian, D. Pardo, M. Neunert, and J. Buchli. Fast trajectory

optimization for legged robots using vertex-based zmp constraints. *IEEE Robotics and Automation Letters*, 2(4):2201–2208, Oct 2017.

[88] Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

[89] Taoyuanmin Zhu, Joshua Hooks, and Dennis Hong. Design, modeling, and analysis of a liquid cooled proprioceptive actuator for legged robots. In *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2019.