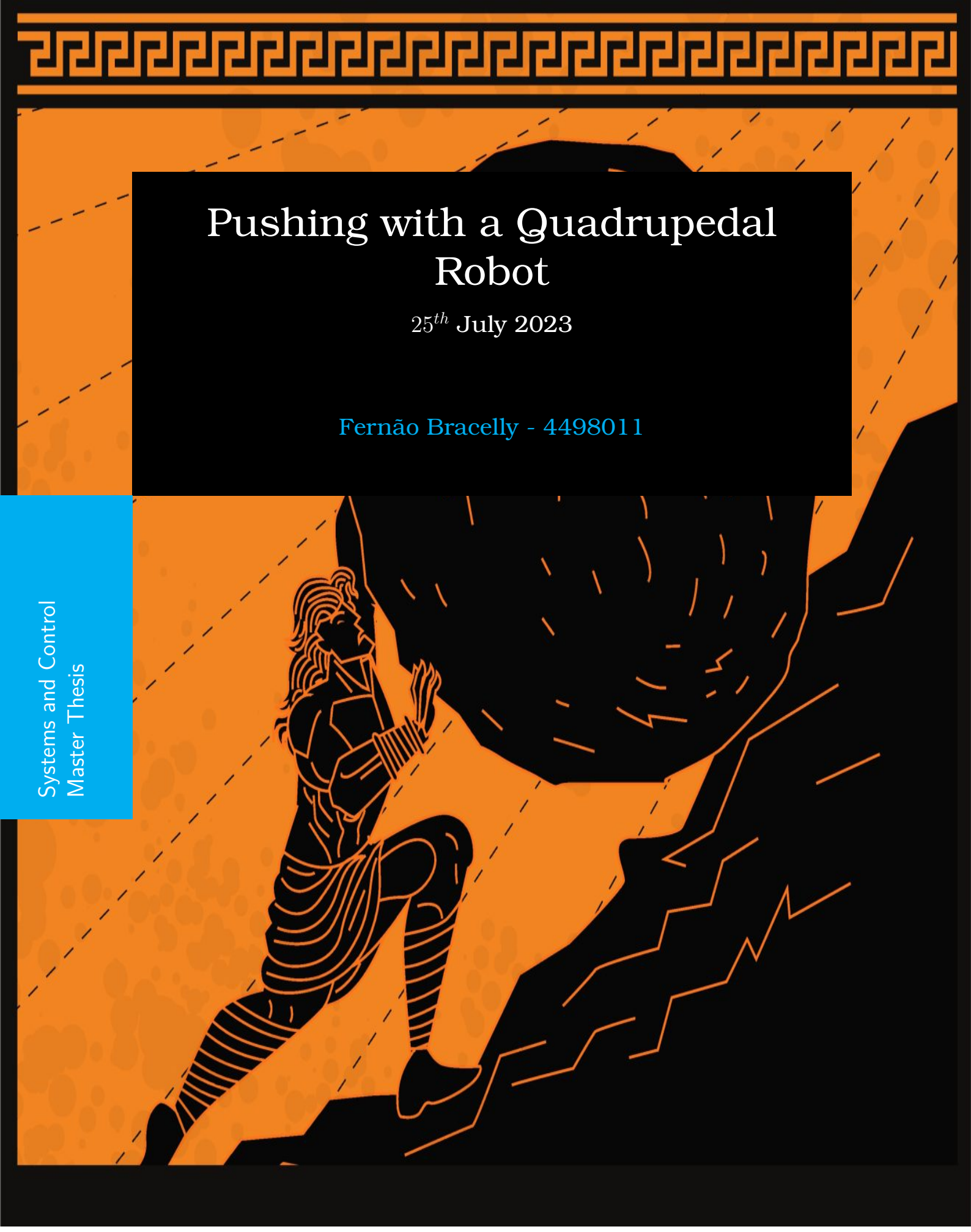
Pushing with a Quadrupedal Robot

25^{th} July 2023

Fernão Bracelly - 4498011

Systems and Control
Master Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Pushing with a Quadrupedal Robot

A proof of concept regarding stable pushing by a quadrupedal robot

Master thesis

Fernão Bracelly - 4498011

25^{th} July 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Pushing with a Quadrupedal Robot

by

Fernão Bracelly

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: 25^{th} July 2023

Supervisor(s):

Prof.dr.ir. Bart De Schutter

Dr.ir. Joris Sijs

Dr. Cosimo Della Santina

Reader(s):

Dr. Koty McAllister

Abstract

Quadrupedal robots possess the ability to move freely in the world and perform a variety of actions that would be unsafe or impractical for humans to perform. In the SNOW project, a quadrupedal robot is tasked with aiding firefighters in rescue missions during house fires by locating humans and assessing their health. Pushing away obstacles that cannot be circumvented otherwise is one of the many capabilities a quadrupedal robot should possess to be of most use in such missions. We develop a proof of concept by solving two problems sequentially: which stance to take on prior to the push and how to perform the push.

The process of stance selection starts with generating a certain amount of stances. Stances are generated starting from a preselected stance appropriate to the goal location and deviating from the 12 joint angles with a normal distribution. All generated stances are ran through a number of filters, which rely on solving for the forward and inverse kinematics of the robot. These filters check if the initial position is sensible and balanced and if the projected final position is close to the goal and balanced. The inverse kinematics are solved using Adaptive-Network-Based Fuzzy Inference Systems (ANFIS), which results in accurate estimations within a time frame that can be used in real-time applications. The final stance is selected by comparing the total displacement of all joint angles per stance, where the lowest total displacement is considered optimal.

The push is controlled by a nonlinear model predictive controller. We strive for a stable push, where the contact between the pusher and the object sticks, by keeping the movement of the end-effector within the motion cone. The motion cone denotes all twists the object can have without slipping at the contact with the pusher and is constructed using the limit surface to model the interaction between the object and the support surface and the generalized friction cone to model the interaction between the pusher and the object. We find that the motion cone predicts stick and slip with an accuracy slightly higher than 80%. Our controller steers accurately to all goals that lie within the motion cone and moves the object with a twist on the edge of the motion cone if the goal location lies outside of the motion cone. The robot remains balanced throughout the pushing motion in the vast majority of cases, but is more at risk of tipping over when pushing heavier objects.

Table of Contents

List of Figures

List of Figures

List of Tables

Acknowledgements

I would like to express my appreciation towards some of the people that aided me most during the time I spent working on my thesis. First of all, I would like to thank dr. ir. Joris Sijs for his role as a supervisor and bearing with me for longer than most others probably would have. I always appreciated the tone of our talks and the input he provided was vital to the completion of this thesis. I would also like to thank dr. Cosimo Della Santina. Though his involvement in this thesis was shorter, he challenged me on some of my work on stance selection and helped me in developing a more efficient approach regarding this topic. I would be remiss if I didn't express my gratitude towards dr. Leandro de Souza Rosa and ir. Hugo Loopik for helping me operate the KUKA robotic arm, as I would have likely remained clueless without them.

Lastly, I must express my thanks to my family and friends for their encouragement and patience during the time I spent working on my thesis, and, of course, for making my life as a student so much more fun and worthwhile than it would have been without them.

Chapter 1 - Introduction

The field of robotics has long since moved away from its confines of performing a limited number of tasks within controlled environments. Whilst robots remain invaluable in classical applications such as industrial manufacturing, the integration of robots into day-to-day life working alongside humans is rapidly becoming a more familiar sight. The development of robots aiding humans in tasks in the open world is one that has made large strides in recent history and this development only seems to pick up in speed [6, 7].

If we want robots to operate in unpredictable environments, they should be equipped with the abilities to do as such. One skill that can be viewed as essential to function well in the real world is the ability to manipulate various objects. When discussing object manipulation, typically, the distinction between prehensile and non-prehensile manipulation is made, where the former involves grasping an object and the latter does not. Grasping is not always a possibility, as not every object is appropriate for any gripper and not every robot is equipped with a gripper. In such cases, the ability to manipulate objects by pushing them is an attractive prospect. Pushing is a complex task to perform for a robot, due to the nonlinear dynamics and unknown friction forces that dictate the planning of a push, but one that has seen many different approaches and successes in the field of robotics [8].

TNO is working on a project titled SNOW (safe autonomous system operating in the open world) that tries to tackle the difficulty of having an autonomous quadrupedal robot operating in the free world, planning its actions in real time. Among the envisioned capabilities of the robot is the ability to push away objects. In the following text we will provide context for the SNOW project and the role of this thesis within the project. In Section 1-1 we go over our use case, detailing what the SNOW project sets out to achieve. In Sections 1-2 and 1-3 we conclude this Chapter by looking into what we want to achieve with this thesis and then how the rest of the thesis is structured, respectively.

1-1 Use Case

The autonomous robot that is used for the SNOW mission is developed by Boston Dynamics, and named Spot. It is a quadrupedal, canine-like robot that is commercially available and has an openly available software development kit. The fact that the robot's software can be altered and extended upon freely is what allow the SNOW project to exist. An image of Boston Dynamics' Spot can be found in Figure 1-1. Boston Dynamics also provides additional hardware that Spot can equip; TNO has equipped Spot with a pan-tilt-zoom camera and a lidar, to further enhance its capabilities of observing the world.

The mission of the SNOW project is to develop several software modules that allow Spot to assist firefighters in locating people during fires and assessing the victims' current health and abilities. In any situation where it is too dangerous for firefighters to directly enter an area of a building, Spot can be deployed instead.

Many challenges are to be overcome for such an idea to become a reality. From effectively communicating with the firefighters, to moving around the house in an efficient manner, there are a great many problems one must consider when tackling this project. One such problem is the ability to manage physical obstacles, obstructing the path of the robot.

We can sketch out a typical mission that the quadrupedal robot might have to take on, to get a better understanding of what falls within the scope of our thesis. Figure 1-2 shows a map

Figure 1-1: Boston Dynamics' robot dog, Spot, which is to be used in the SNOW project.

of a house Spot has been ordered to search and it shows its current situation: the kitchen and living room are on fire, with smoke having filled the living room as well. The house is inhabited by a family and it is unknown whether or not the family is still present in the house.

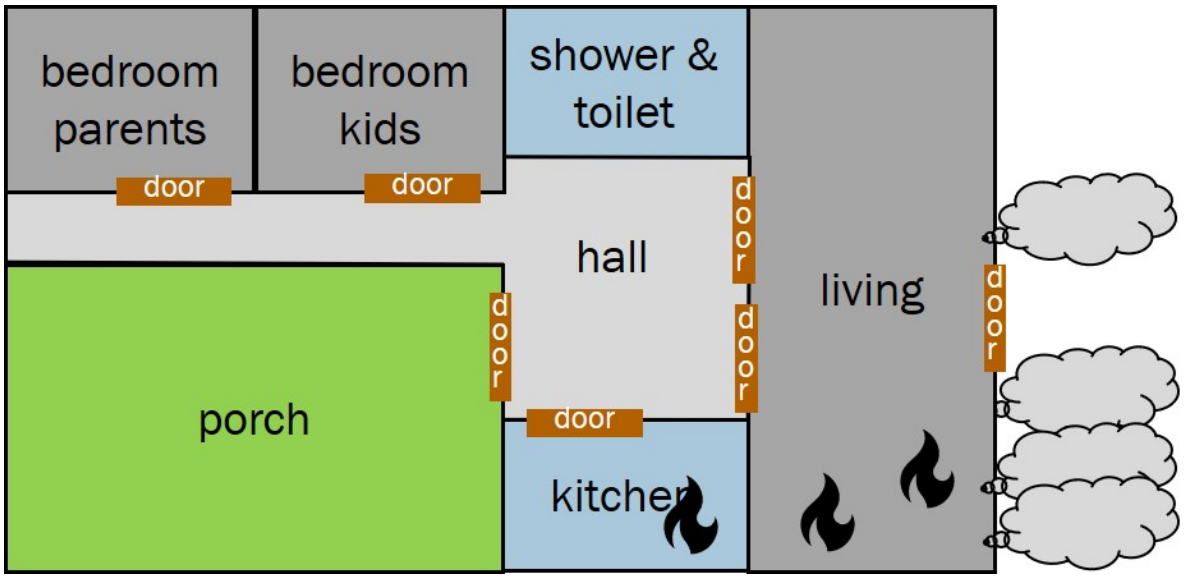


Figure 1-2: Map of the house that Spot is to search. As can be seen, the kitchen and living room are on fire and the living room is filled with smoke as well.

In such a scenario it is certainly not safe to assume that the path to every room is clear of debris or an object obscuring the way, nor can we assume that the path from the robot to one of the family members can be walked without encountering a similar obstacle. The scenario at hand simply has too much inherent uncertainty to not account for such cases.

There are essentially two things that can be done when encountering an object that stands

between you and the location you need to go to: circumvent it or move it out of the way. The former will typically be the preferred method of dealing with such an object as it is faster and quadrupedal robots are developed to be agile and maneuverable, but there are cases where this is not a possibility, for example if the middle of a hallway is blocked by a tall obstacle and on both sides of the hallway there is not enough room for Spot to fit through. In such a case, pushing the obstacle out of the way is the only resource it has left. Finding a reliable way of pushing aside various objects is what we will concern ourselves with in this thesis. This functionality is a minor one in the grand scheme of things. That being said, however, the addition of functionalities that might prove life-saving given this scenario and the more capabilities Spot has to aide in a house fire, the better.

1-2 Problem Formulation

The problem we concern ourselves with in this thesis, is that of non-prehensile object manipulation by a quadrupedal robot in house fires. Tackling this problem in its entirety would expand the scope of our thesis to be unfeasible. Rather, we should focus on forming a coherent problem formulation that addresses what is most important: designing a controller that allows a quadrupedal robot to push away objects.

Having a quadrupedal robot push away objects with one of its own limbs is unexplored in scientific literature, meaning we have develop our own approach. We identify two separate problems that have to be solved to come to a viable controller. These problems are:

- What stance should the quadrupedal robot take on prior to the push?
- How should the push be performed?

We have to disregard multiple notable aspects of our use case to tackle this problem. Most importantly, we do not consider that the robot is confined to a specific area and we disregard the hazards that might threaten the robot during the push. We argue that it is necessary to abstract away these aspects to keep the scope of this thesis manageable. We rely on other assumptions, which are named in Chapters 2 and 3, but they are not as fundamental in nature.

We can articulate the goal of this thesis:

▷ **Design a controller for a quadrupedal robot that can push away various objects obstructing its path.**

This thesis is to be seen as a proof of concept. We do not deliver a controller that is fully functional in the real world, but rather a controller that addresses all necessary functionalities for a quadrupedal robot to perform a push.

1-3 Thesis Structure

Our method of control consists of two separate components: one that determines an appropriate stance and one that controls the push. Selecting a stance precedes the push and has significant influence on the push itself. The stance of the robot and the planning of the push both rely on the goal we wish to push towards and the perceived properties of the object we

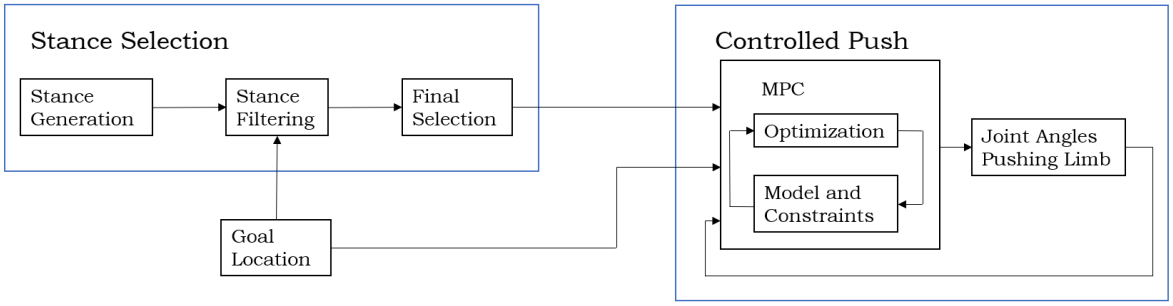


Figure 1-3: High-level overview of the control strategy this thesis describes.

wish to push. A high-level depiction of the control strategy that we will work out in this thesis is shown in Figure 1-3.

In Chapter 2 we discuss the topic of stance selection. Here we first address the necessary assumptions to come to be able to generate stances for our use case, after which we discuss the forward and inverse kinematics, which are used to determine the starting stance and estimate the stance after the push has been executed. Then we go over a number of filters which parse out whether a stance is practicable or not and we discuss how a final choice is made through the chosen optimality criterion.

The topic of Chapter 3 is the control of the push and the modeling framework we use. First we discuss the mechanics of pushing, outlining how we model pushing and how it is to be integrated in our control strategy. Then we validate this approach through an experiment. Finally, we address the controller and its performance in simulation.

In the final Chapter, Chapter 4, we look at the contents of the previous Chapters and discuss our choices. We also look into what future work is most pressing to come to a more complete controller, and conclude our thesis with a summary and a final statement.

Chapter 2 - Stance Selection

We identified in Chapter 1, that the problem we are tasked with solving can be split into two different, smaller problems, namely determining an appropriate stance that the quadruped should take on in order to successfully push the object to its goal and the actual push of said object towards the goal. This Chapter concerns itself with the former problem: investigating how we can come to a stance that is appropriate to the situation. As this problem is not tackled in scientific literature, we come to a solution to this problem by combining several more common techniques to solve it. We will also see, in Chapter 3, that the stance selection has direct implications on how successful the subsequent push will be. This Chapter goes through all the steps needed to come to a stance that can reasonably expected to form the basis of a successful push, and Chapter 3 revisits the topic of stance selection to investigate its influence during the push.

The basic principle of determining the stance is as follows: we generate a number of stances, based on twelve angles (three per limb), and every generated stance is assessed utilizing a number of tests, which identify whether the stance is eligible or not, before selecting the most appropriate remaining stance based on an optimality criterion. We can generate these stances in different ways, which leads us to different end results, even if the set of tests the generated stances go through remain the same.

This Chapter firstly addresses the assumptions that are made that are relevant to the our problem. Then we go over the tests that are used during the selection process. These tests rely on solving the robot's forward and inverse kinematics. In the selection process we make use of three separate adaptive neuro-fuzzy inference systems (ANFIS) to solve the inverse kinematics of the robot, a topic that will be addressed separately. We also discuss and justify the filters and optimality criterion that we employ. Section 2-1 goes over the assumptions we have made, Section 2-2 goes over all the steps required to get from a number of generated stances to a final choice. Here, Sections 2-2-1 and 2-2-2 discuss the way we handle the forward kinematics and inverse kinematics of the robot, respectively. In Section 2-2-3 we investigate the optimality criterion. Finally, in Section 2-3, we summarize what we have discussed in this Chapter and come to a conclusion. This will leave us with an understanding of how our stance is selected prior to the pushing of the object and the guarantee that the selected stance is the best option of all generated stances based on the optimality criterion.

2-1 Assumptions

Before we make any meaningful decisions regarding which generated positions we discard and which we deem suitable, we have to set some boundary conditions and make reasonable assumptions about both the quadrupedal robot and its environment.

We will first discuss go over the robot parameters. These are not based directly on any particular robot, but rather chosen to be representative of a generic, smaller quadrupedal robot. The performance of the stance selection, and later the pushing of the object, does not hinge on these parameters being exactly the way they are represented here; changing the parameters to be in congruence with a particular quadrupedal robot should not influence the quality of the stance selection procedure.

We also must make assumptions on the knowledge we have of the system. As can be inferred from Chapter 1, we work on a flat plane with no other obstacles other than the object we

wish to push. Furthermore, we assume to have full knowledge of the relevant properties of the object we wish to push. The validity of these assumptions is discussed in Chapter 4.

The set of assumptions relevant to this Chapter are:

- Size and mass of the body: The body is assumed to be a cuboid. Its dimensions are 1 meter in length, 0.5 meters in width and 0.2 meters in height, with a mass of 20 kilograms and a constant density.
- Size and mass of the limbs: The limbs are assumed to be cylindrical. The robot's thighs are 0.5 meters in length and have a radius of 0.075 meters, while the robot's shins are 0.4 meters in length and have a radius of 0.0375 meters. Each thigh has a mass of 1.25 kilograms and each shin a mass of 0.25 kilograms for a total mass of 1.5 kilograms per leg, and both the thighs and the limbs have the same constant density.
- Movement of the limbs: Each leg can perform three different rotations, two at the top joint (the hip) and one at the bottom joint (the knee). At the hip joints, the robot is capable of adduction/abduction, that is motion towards and away from the body's midline respectively, and flexion/extension, movements that respectively decrease and increase the angle between the body and the thigh. At the knee joint, the quadrupedal robot is only capable of flexion and extension, expressed as the angle between the thigh and the shin. The limits to these movements are as follows:
  - Adduction and abduction of the hip: -45 degrees to 45 degrees from vertical. The maximum rotation speed is 20 deg/s in both directions.
  - Flexion and extension of the hip: -150 degrees to 40 degrees from vertical. The maximum rotation speed is 20 deg/s in both directions.
  - Flexion and extension of the knee: 20 degrees to 160 degrees from straight. The maximum rotation speed is 20 deg/s in both directions.
- We assume that the angles of the joints are known at all times and we can achieve all joint angles that are within the set bounds.
- The robot exclusively operates on flat, level ground.
- There exists a known goal location, which is the location the robot wishes to move its arm towards.
- The mass, dimensions and coefficient of friction of the object that is to be pushed are known.

The values chosen for the robot's parameters are in line with smaller quadrupedal robots, such as Boston Dynamics' Spot robot and Deep Robotics' Jueying Mini [1, 9].

As far as naming conventions go, we will refer to the legs of the robot as front and hind and left and right legs. The left front leg is referred to as leg 1, the right front leg is leg 2, the left hind leg is leg 3 and the right hind leg is leg 4. The robot's body frame is defined as depicted in Figure 2-1, where the x-axis is aligned with the body lengthwise, the y-axis is aligned with the body from right to left, and the z-axis is aligned with the body from down to up. We

choose the geometric center of the main body of the robot to be the origin of the coordinate system of the body frame. Though it is more conventional to take the center of mass as the origin of the body frame, our choice is sensible in our case, as this means that the origin does not shift when the robot moves its legs.

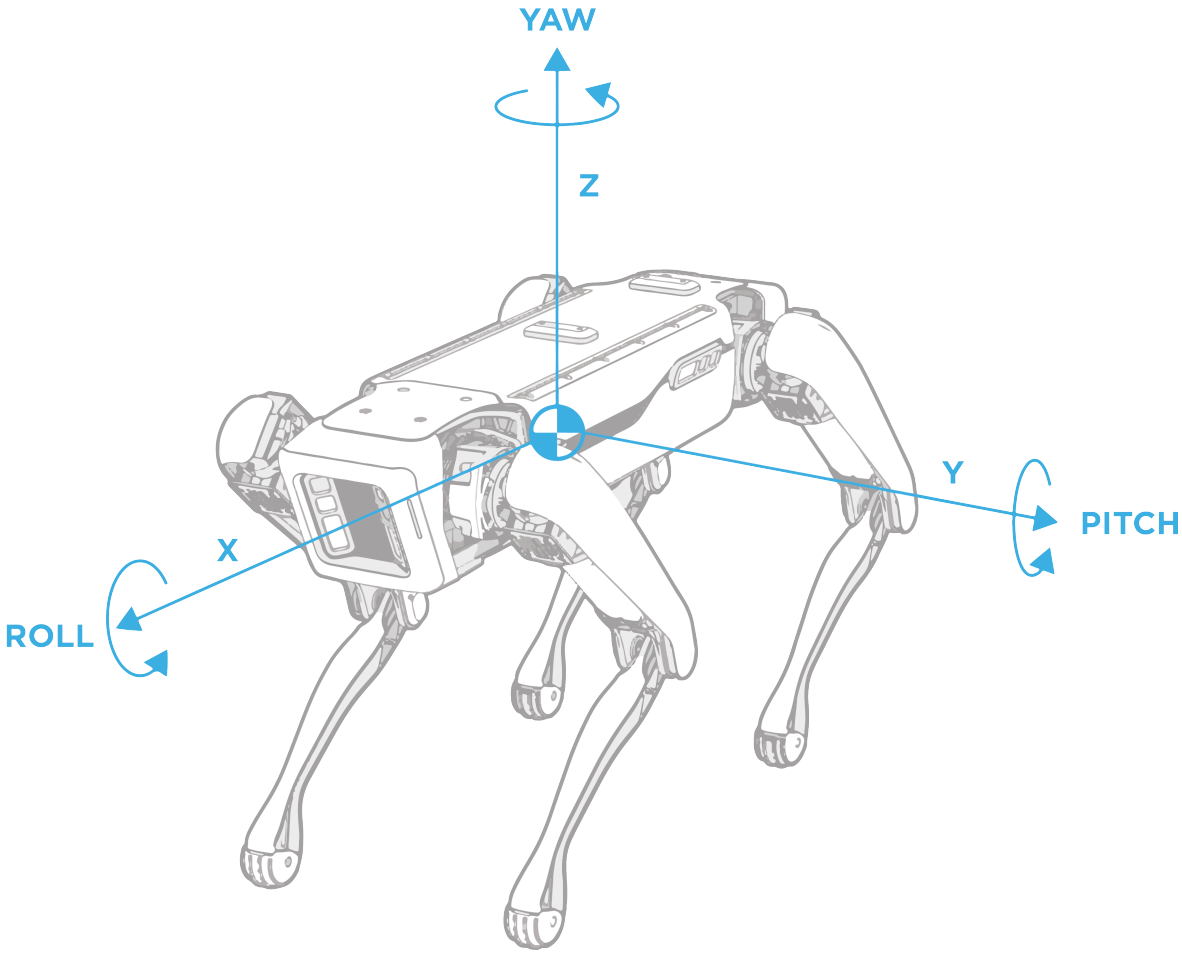


Figure 2-1: A depiction of the body frame on Boston Dynamics' Spot. From [1].

With these basic assumptions and conventions outlined, we can look into stance generation and the selection process.

2-2 Stance Filtering

The quadruped has twelve degrees of freedom, three per leg, allowing it to take on a wide range of stances. If we were to randomly generate a large multitude of stances, however, we would quickly find out that the majority of these stances are unusable for one reason or another. For example, some of these stances would be unbalanced, causing the robot to tip over, and others would not be able to reach the target we are pushing towards.

In order to know which stances to filter out and which to keep, we must first look into the kinematics of the quadrupedal robot. When the forward kinematics have been established, we must also look into the inverse kinematics, as the inverse kinematics are essential to scouting out whether or not the pushing limb has the ability to reach the intended pushing goal or not.

When both the forward kinematics as well as the inverse kinematics have been discussed, we will look at the filters we apply to get to a sensible stance that can be used during pushing.

2-2-1 Forward Kinematics

An understanding of the robot's forward kinematics allows us to determine the position of all end-effectors, in our case the feet of the robot, from the robot's joint angles. Figure 2-2 depicts a quadrupedal robot in such a way that all joint rotations are easily readable. The hip joint can be seen as two separate joints, one rolling (adduction/abduction) and one pitching (flexion/extension) joint, and the knee can be seen as a singular pitching joint. It should be said that, while the joint rotations match with the body frame in Figure 2-2, this is not necessarily the case. If one leg performs adduction, for example, the pitching joints no longer align with the y-axis of the body frame, which is something we have to consider. In all future references to the robot, for convenience, we will depict and represent the robot as a frame made up of links, instead of a more complicated depiction.

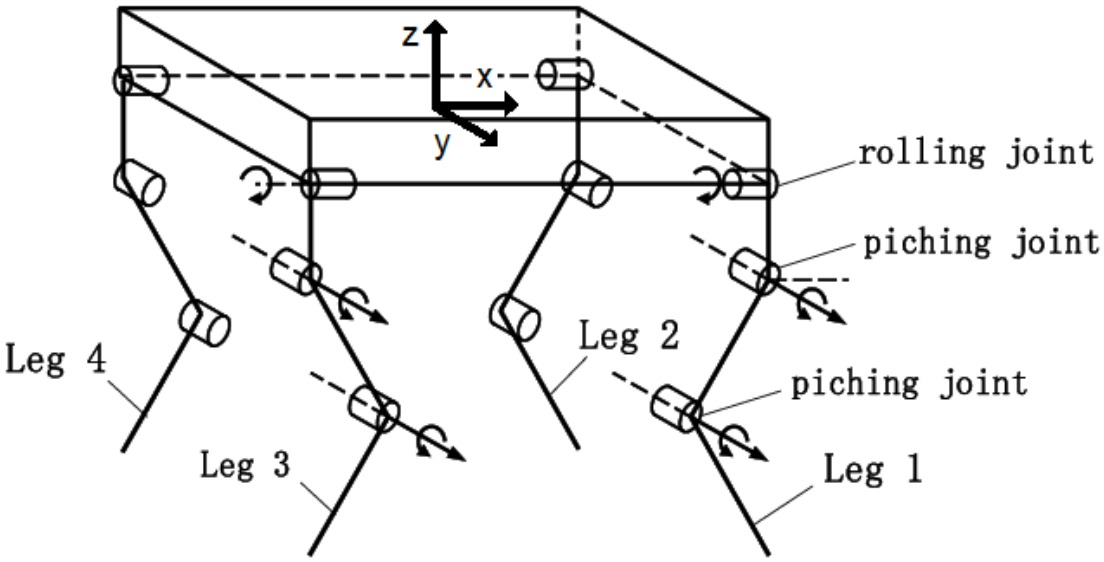


Figure 2-2: Schematic representation of the quadrupedal robot and its leg joints. Adapted from [2].

Our goal in this section is twofold: we want to parse out the position of each foot based on the joint angles and we want to determine the world frame, based on the feet that are in contact with the ground. The first step to determining the position of each foot is to examine how joint rotations change the orientation of a leg.

There are two modes of rotation that are of interest to us when it comes to the rotation of the leg joints: roll and pitch. The rotation matrix R_r associated with roll, describing a counterclockwise rotation of θ_r about the x-axis is:

$$R_r(\theta_r) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_r & -\sin\theta_r \\ 0 & \sin\theta_r & \cos\theta_r \end{bmatrix} \tag{2-1}$$

And the rotation matrix R_p associated with pitch, describing a counterclockwise rotation of θ_p about the y-axis is:

$$R_p(\theta_p) = \begin{bmatrix} \cos\theta_p & 0 & \sin\theta_p \\ 0 & 1 & 0 \\ -\sin\theta_p & 0 & \cos\theta_p \end{bmatrix} \tag{2-2}$$

We must also establish how each angle is defined. We will call the angle of adduction/abduction at the hip θ_{ah_i}, where i denotes the leg the angle refers to, we will call the angle of flexion/extension at the hip θ_{fh_i}, and the angle of flexion/extension at the knee θ_{fk_i}. Angle θ_{ah_i} is 0 degrees when there is no adduction or abduction and the thigh is pointed in the negative z-direction with respect to the body frame. As for angle θ_{fh_i}, it is 0 degrees when there is no flexion or extension and the thigh is pointed downwards with respect to the local joint frame. Finally, angle θ_{fk_i}, which is 0 degrees when the angle between the thigh and shin in the joint frame is 0. Do note that this angle, due the assumptions outlined in Section 2-1, is never 0 for the quadrupedal robot we are considering; the thigh and shin are always at an angle in the joint frame.

The position of knee i in the body frame, P_{k_i}, can be calculated as follows:

$$P_{k_i} = d_i + R_p(\theta_{fh_i})R_r(\theta_{ah_i}) \begin{bmatrix} 0 \\ 0 \\ -l_{thigh} \end{bmatrix} \tag{2-3}$$

Where d_i is the distance between hip i and the origin of the body frame, and l_{thigh} is the length of the thigh.

The position of foot i, P_{f_i} can then be calculated to be:

$$P_{f_i} = P_{k_i} + R_r(\theta_{ah_i})R_p(\theta_{fk_i} + \theta_{fh_i}) \begin{bmatrix} 0 \\ 0 \\ -l_{shin} \end{bmatrix} \tag{2-4}$$

Where l_{shin} is the length of the shin.

Given the angles of the joints, we can calculate the position of each knee and each foot, using equations 2-3 and 2-4, respectively. This, however, is not sufficient to start the procedure of selecting positions, because we do not know the robot's orientation in the world frame, nor do we know which feet touch the ground. We should go over the procedure we use to determine these facts.

There is a limited amount of possibilities when it comes to determining how many feet are in touch with the ground. Either all four feet touch the ground, or any configuration of three feet touch the ground. Our procedure assumes that three feet are touching the ground, and that all four possible configurations, $(1, 2, 3)$, $(1, 2, 4)$, $(1, 3, 4)$, $(2, 3, 4)$, are eligible. It then goes through each configuration, finds the transformation matrix to go from the body frame to the world frame, a process we will describe shortly, and we then ascertain whether or not the standing feet are positioned lower than the foot we assume to be raised in the world frame. There exist two possible outcomes for this test. We may find that the raised foot has an equal

height in the world frame in all four configurations, meaning that the robot is standing on all four of its feet, or we may find that only one configuration where the foot that is assumed to be raised is indeed located higher in the z-direction of the world frame, in which case the robot is standing on three feet and that configuration is the correct one. We will go over this of this process in more detail, showcasing how the transformation matrix is constructed and employed.

The three feet that we assume to be touching the ground span a triangular area, which is part of the level ground the robot is standing on. We use this triangular area to come to the world frame. We can find the normal vector perpendicular to said area, vector z_{ground}, by taking the cross product of two vectors that lie on the ground plane. We can do this in the following manner:

$$z_{ground} = \frac{(c_1 - c_2) \times (c_1 - c_3)}{|(c_1 - c_2) \times (c_1 - c_3)|} \tag{2-5}$$

Where c_1, c_2, and c_3 are the three contact points we assume the robot is making with the ground with its feet, expressed in the body frame and $\times$ denotes the cross product. It should be noted that if the third component of z_{ground} is negative, we multiply the vector by -1, so that the normal vector always points upward from the ground.

An observation we can make is that, in our use case, there can be a pitch and roll between the robot and the ground, but there is no use for a yaw angle. As we are working on flat, level ground, the yaw angle between the ground frame and body frame is inconsequential. We can set the yaw angle to zero, by creating an axis, in our case the y-axis of the ground frame, that is a projection of the y-axis of the body frame onto the level ground. We can create this unit vector as follows:

$$y_{ground} = \frac{\begin{bmatrix}0\\1\\0\end{bmatrix} - \left(\begin{bmatrix}0\\1\\0\end{bmatrix} \cdot z_{ground}\right) z_{ground}}{\left|\begin{bmatrix}0\\1\\0\end{bmatrix} - \left(\begin{bmatrix}0\\1\\0\end{bmatrix} \cdot z_{ground}\right) z_{ground}\right|} \tag{2-6}$$

Here $\cdot$ denotes the dot product.

Finally, we can create the third base vector of the world frame, x_{ground}, by creating a vector perpendicular to z_{ground} and y_{ground}. This is done as follows:

$$x_{ground} = \frac{z_{ground} \times y_{ground}}{|z_{ground} \times y_{ground}|} \tag{2-7}$$

It should be noted, that we, in a similar fashion as what we did with z_{ground}, multiply vector x_{ground} with -1 if its first component is negative, to create less confusing ground axes.

The vectors that we create in equations 2-5, 2-6 and 2-7 form the basis of rotation matrix R_{gb} that can be used to transform ground frame coordinates to body frame coordinates, but we

are interested in the inverse: going from body frame coordinates to ground frame coordinates. This can be done by using the transpose (or inverse) of R_{gb}, namely R_{bg}:

$$R_{bg} = \begin{bmatrix} x_{ground} & y_{ground} & z_{ground} \end{bmatrix}^T \tag{2-8}$$

Now that we can go from the body frame to the world frame, we are yet to see which feet the robot is actually standing on. This can be done straightforwardly; we check for each possible combination of contact feet, whether the contact feet have a z-coordinate below or equal to the foot of which we assume that is not making contact to the ground, in the ground frame. This either leaves us with one option, or four. In the former case, the lone option tells us which three feet make contact to the ground, and in the latter all four feet make contact.

We now have the capacity to formulate the robot's stance in the world frame given the joint angles. Not every set of joint angles will produce a stance that is feasible or practicable. Some stances might, for instance, be inherently unbalanced and cause the robot to tip over, other stances might have one or several knees touching the ground, rather than three or four feet. The topic of filtering out stances that are not usable for us is discussed in Section 2-2-3. Before we can filter out stances, however, we have to discuss the issue of inverse kinematics.

2-2-2 Inverse Kinematics

The application of forward kinematics is explicitly necessary for generating stances and selecting the ones that might be used. The understanding of the robot's inverse kinematics is important in our use case as well, though it is not needed for the generation of stances. Rather, the inverse kinematics gives us an idea of how the robot might end up once the pushing motion has been completed. Inverse kinematics will help us examine whether or not the position we wish to reach lies within the range of motion of the pushing limb and, consequently, whether the final position of the pushing limb keeps the robot in a balanced position or not.

All methods of solving the inverse kinematics problem take a given position of an end-effector and produce the angles needed to get to said position. This problem is much less straightforward than solving for the robot's forward kinematics. A complicating matter in inverse kinematics is the fact that any given single point in space might have a multitude of configurations that match it, one singular configuration or none at all, an issue not present in forward kinematics [10]. There exist many methods that solve the inverse kinematics problem, which can be grouped into three different approaches: closed form, numeric and iterative approaches. Typically, both closed form, which cannot solve for all robot manipulators, and numeric approaches have a heavy computational load, making them unideal choices in live applications of inverse kinematics. Iterative approaches, on the other hand, are a more appropriate choice for live applications, as their computational load is typically much lower [11]. One such iterative approach makes use of Adaptive-Network-Based Fuzzy Inference Systems (ANFIS), also referred to as Adaptive Neuro Fuzzy Inference Systems.

We have opted to solve the inverse kinematics using ANFIS, as its real-time capabilities might prove to be of vital importance in our use case. In the following we will describe how ANFIS networks work and how we apply them in our use case, after which we display how it performs at solving the inverse kinematics.

ANFIS

ANFIS networks were first introduced by Jang in 1993 [4], where he describes his method of integrating Fuzzy Inference Systems (FIS) into the framework of artificial neural networks (ANN). The operations that an ANFIS network performs can best be explained by looking at the architecture of the network and working through it layer by layer. Figure 2-3 displays the network architecture of an ANFIS network with two inputs, x and y, a fuzzification layer, then consecutively a product layer, a normilization layer, a defuzzification layer and a summation layer. The learning process applies to the fuzzification and defuzzification layers, as we will see. This is the standard network architecture for ANFIS networks, though the amount of inputs, and number of membership functions and fuzzy rules may differ. The network we describe is referred to as a type 3 ANFIS network.

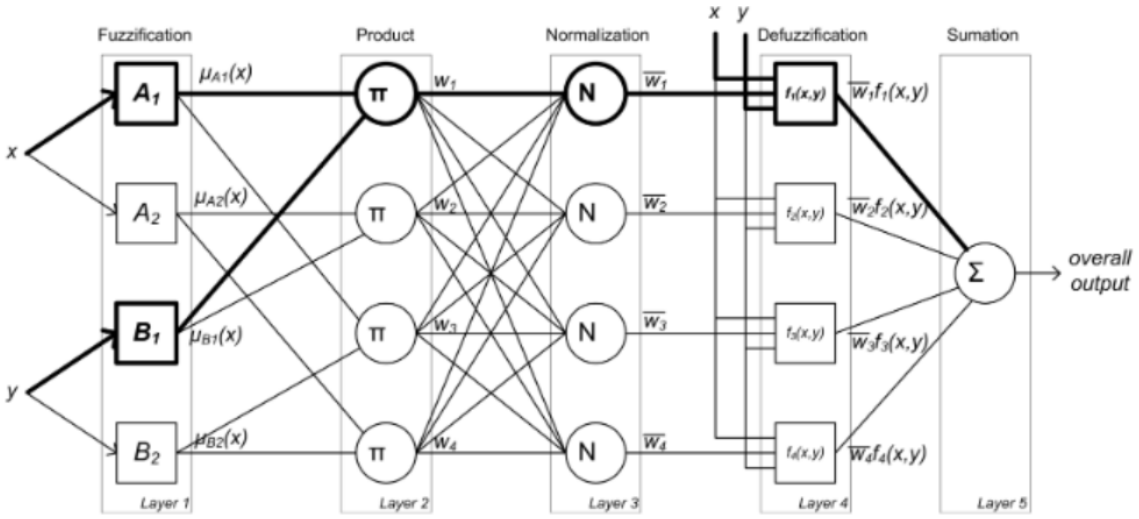


Figure 2-3: ANFIS network architecture. From [3].

The first layer, the fuzzification layer, receives each input and puts them through a number of membership functions, in our example, shown in Figure 2-3, two per input. These membership functions are parameterized by so-called premise parameters which change during the learning process. There exist a number of functions that are used often as membership function; we will look at the generalized bell function:

$$\mu_{A_i}(x) = \frac{1}{1 + [(\frac{x - c_i}{a_i})^2]^{b_i}} \tag{2-9}$$

Where A_i is a quantifier, often associated with a descriptive, linguistic label. The value of the membership function lies between 0 and 1 and represents the degree of truth or degree of membership, where a value closer to 1 indicates the input variable is closely associated to the class A_i represents. Values a_i, b_i and c_i are the premise parameters that are learned during training. Figure 2-4 depicts the generalized bell function and how the premise parameters shape it.

In the second layer, the product layer, the values of the corresponding membership functions are multiplied with one another. The output of the nodes at this layer is called the firing

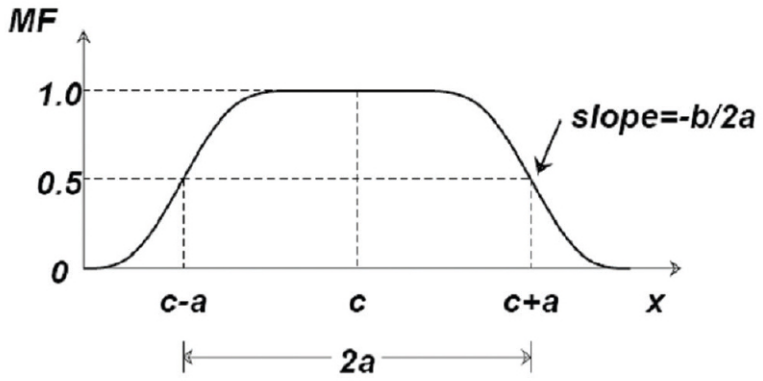


Figure 2-4: Shape of a generalized bell membership function. From [4].

strength. For example, the operation at the product layer node corresponding to the blackened edges in Figure 2-3 is:

$$w_1 = \mu_{A_1}(x) \cdot \mu_{B_1}(y)$$

The third layer takes the firing strength of the associated node and normalizes it, by dividing the firing strength by the sum of all rules' firing strengths. Thus the output, called the normalized firing strength, $\bar{w_1}$, as seen in Figure 2-3 for instance, is calculated as follows:

$$\bar{w_1} = \frac{w_1}{\Sigma_{i=1}^{4} w_i} = \frac{w_1}{w_1 + w_2 + w_3 + w_4}$$

In the defuzzification layer, the normalized firing strength is multiplied by a function of the inputs. This multiplication, given that there exist two input variables in our example, x and y, is:

$$\bar{w_i} f_i(x, y) = \bar{w_i}(p_i x + q_i y + r_i)$$

Where p_i, q_i and r_i are the consequent parameters, whose value changes during the learning process, like the premise parameters we encountered in the first layer.

In the final layer, all outputs of the defuzzification layer are summed together to produce the final output of the ANFIS network. In our example that output is $\Sigma_{i=1}^{4} \bar{w_i} f_i(x, y)$.

ANFIS networks adapt the premise parameters of the fuzzification layer and the consequent parameters of the defuzzification layer through supervised learning, typically using a hybrid learning algorithm. Here, the premise parameters are learned using the backpropagation algorithm and the least-squares error method to tune the consequent parameters [12].

With the basics of ANFIS networks covered, we can look into the implementation in our use case and its performance at solving the inverse kinematics problem.

Solving Inverse Kinematics using ANFIS

Our implementation of ANFIS is largely based on a paper by Duka [3], where three separate ANFIS networks are trained using not only generated positions of the end-effector but also supplying the network with the orientation of the robot arm. In his paper, Duka defines the end-effector orientation of the arm as the sum of the angles each link, of which there are three, makes. The described robot arm only has a x- and y-coordinate, which is different from our case. Still, as we will see, this use of arm orientation is something we can use to feed into the training data of our ANFIS networks, to solve the inverse kinematics of the quadruped's pushing limb to a satisfactory degree.

To find joint angles, given the end-effector position and our substitute for the end-effector orientation used in [3], we use three ANFIS networks with the same architecture, one each for estimating θ_{ah_i}, θ_{fh_i}, and θ_{fk_i}, as defined in Section 2-2-1. The inputs for each network are the x-, y- and z-coordinate and the θ_{fh_i} angle. The joint angle θ_{fh_i} is our substitute for the orientation in Duka's paper and without it the ANFIS networks perform poorly. Each input has three membership functions, meaning that there are $3^4 = 81$ total fuzzy rules per ANFIS network. There is an obvious problem with using θ_{fh_i} as an input for the ANFIS network, as we do not know what this angle should be after defining the goal we want to push the object to, as opposed to the x-, y- and z-coordinate, which we do know. This problem is addressed with a lookup table; a process which we will describe later on.

The data necessary to train the ANFIS networks is generated using the forward kinematics of the pushing limb. Given the range of motion of each joint described in Section 2-1, we create three vectors containing fifteen values for θ_{ah_i}, θ_{fh_i} and θ_{fk_i} each, where all fifteen angles are evenly spaced apart. From these angles we create $15^3 = 3375$ unique pushing limb configurations and we calculate the end-effector position, storing the value for θ_{fk_i} associated with each end-effector position. The column vectors containing the x-, y-, and z-coordinates and the associated θ_{fk_i} angles are used as training data and the values for θ_{ah_i}, θ_{fh_i} and θ_{fk_i} are supplied so that the three networks can learn to approximate them.

All three ANFIS networks were trained over the course of 150 training epochs, using Matlab's fuzzy logic toolbox. To test the three ANFIS networks and their performance we create three vectors for the joint angles with 30 evenly spaced values each and we calculate the 27000 end-effector positions associated with those angles. These end effector positions and the angle at the knee θ_{fk_i} are fed into the trained ANFIS networks which produce an estimate for the correct joint angles, which are then fed into the forward kinematics to produce an estimate of the end-effector position. We specifically estimate the position of foot 1, the front left foot, in the body frame, but the networks can be applied to any foot and the rotation matrix from equation 2-8 can be used to go from the body to the world frame, so the method is agnostic to which foot we use and how the robot is oriented with respect to the world.

The results of this process can be seen in Figures 2-5 and 2-6. In Figure 2-5 we can see the total prediction error and the error regarding each coordinate, sorted by the magnitude of the error. It is clear that the errors are very small, considering the size of the limb is 0.90 meters. The mean total error is 1.78 mm, where the mean errors of the x-, y- and z-coordinate are 1.01 mm, 0.586 mm, and 1.02 mm respectively. The median total error is 1.47 mm and the median errors of the x-, y- and z-coordinate are 0.683 mm, 0.416 mm, and 0.707 mm. The errors in the x- and z-coordinate are significantly higher than those in the y-coordinate and

the fact that all median errors are approximately 30% lower than the mean errors suggests the existence of larger outliers. Indeed, we can see from Figure 2-5 that there exist errors that are multitudes higher than the mean or median errors. The errors for all coordinates follow a similar trajectory, where most errors lie within the same range, with a small portion that exceeds this range. The outliers of the x- and z-coordinates have a bigger impact than that of the y-coordinate as the errors for these coordinates are larger in general.

The histogram in Figure 2-6, shows how the errors are distributed and showcases that the vast majority of the errors do not rise above 5 mm, and the outliers that do, especially the ones that are considerably higher, are rare.

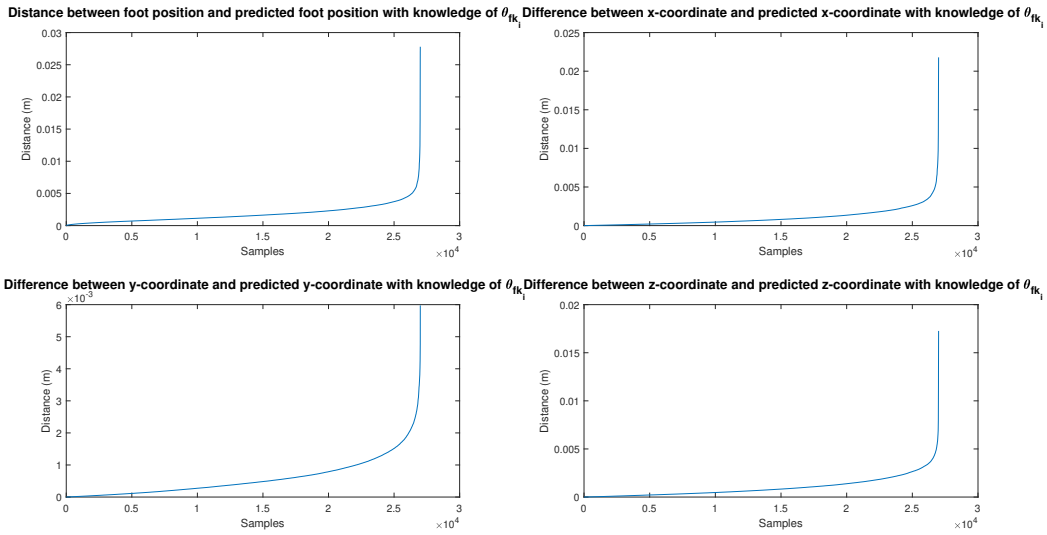


Figure 2-5: Plots of the errors in estimated foot position with perfect knowledge of θ_{fk_i}, sorted by magnitude.

These results are encouraging, but we do need to tackle the fact that we will not have perfect knowledge of the angle θ_{fk_i}, that all three ANFIS networks need as an input. We resolve this problem by using a lookup table, as mentioned earlier. As we have done before, we construct three vectors, one for each joint and calculate the appropriate end-effector position using the forward kinematics. In this thesis, these vectors all have 20 entries, evenly spaced apart. More entries would make for a slower, more accurate estimate, so this size is a trade-off.

We make use of this lookup table to come to a reasonable estimate of what θ_{fk_i} should be. We do not know the angle θ_{fk_i} that should be fed into the ANFIS networks when we try to estimate an end-effector position close to our pushing goal, but we do know the desired x-, y- and z-coordinate. We take the pushing goal and compute the distance between the goal and each of the $20^3 = 8000$ entries in our lookup table. We read out the angle θ_{fk_i} from the entry in the lookup table that lies closest to the desired goal and feed it into the ANFIS networks to estimate the position. The results of this approach can be seen in Figures 2-7 and 2-8.

Figure 2-7 depicts the sorted prediction errors and we can observe several differences between it and Figure 2-5. As is to be expected, the errors have become larger, by one order of magnitude, with the mean total error being 2.13 cm, and the errors of the x-, y- and z-coordinate being 1.30 cm, 3.76 mm, and 1.34 cm respectively. The median total error is 1.66

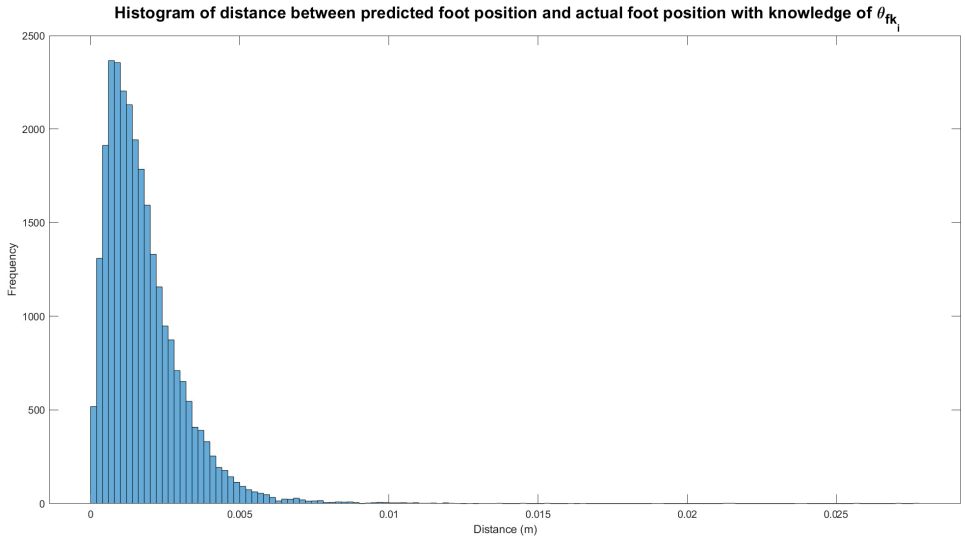


Figure 2-6: Histogram showcasing the error in estimated foot position, with perfect knowledge of θ_{fk_i}.

cm, and the median errors of the x-, y- and z-coordinate are 7.68 mm, 2.12 mm, and 8.59 mm respectively. The differences between the mean and median error for each coordinate has grown to around 40%, suggesting that the amount of outliers is more numerous. We can also see this in the shapes of the plots in Figure 2-7 compared to those in Figure 2-5, as the plots rise earlier and more gradually than they did in the case where we have perfect knowledge of angle θ_{fk_i}.

Figure 2-8 depicts a histogram of the prediction errors for the foot position and it supports the notion that the spread of the errors is less compact than it was when a lookup table was not necessary. A more significant portion of the estimates has larger errors, but the overall performance is still acceptable with the majority of errors not exceeding 5 cm. The average time needed to come to an estimate of the position is 15.3 milliseconds on an Asus FX504G computer, 500 times slower than the 30.5 microseconds it takes with perfect knowledge, but certainly still appropriate for our use case.

2-2-3 Filtering and Optimality

We have solved the forward and inverse kinematics of the robot in Sections 2-2-1 and 2-2-2, so that we may come to robot stances that allow us to push away an object blocking the robot's path. We can generate any number of angles per joint that are within each joint's range of motion, but not all resulting stances are useful. For this reason every generated position is subject to a number of filters that rely on the forward and inverse kinematics, to make sure that we only work with poses that are sensible. In this Section, we will go over all the filters that are employed, discuss the optimality criterion and touch on stance generation.

The first set of filters concern only the forward kinematics of the robot. The first filter each stance goes through, checks whether or not all standing feet are located at a lower z-coordinate than all four of the robot's knees. If this is not the case, the stance is discarded, as it is obviously not acceptable to have the robot stand on its knees.

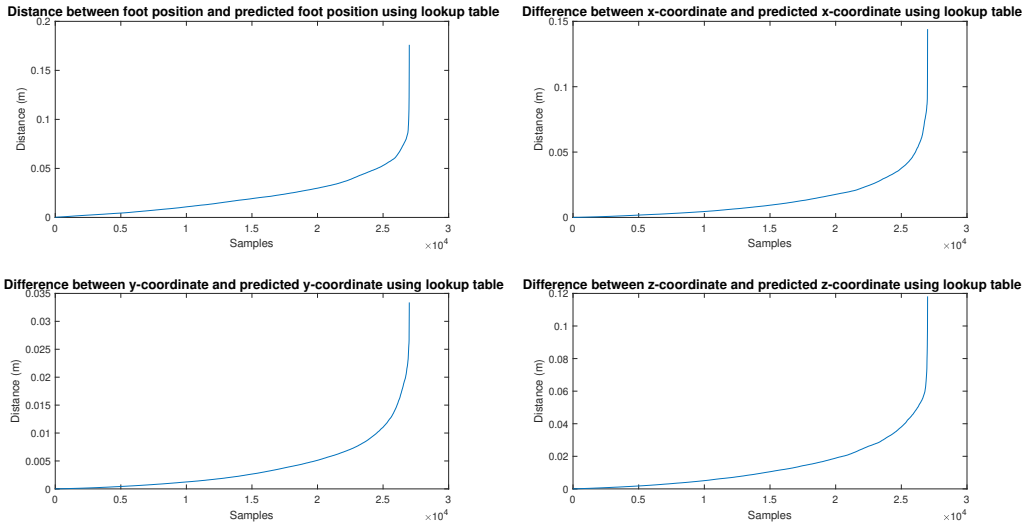


Figure 2-7: Plots of the errors in estimated foot position using a lookup table, sorted by magnitude.

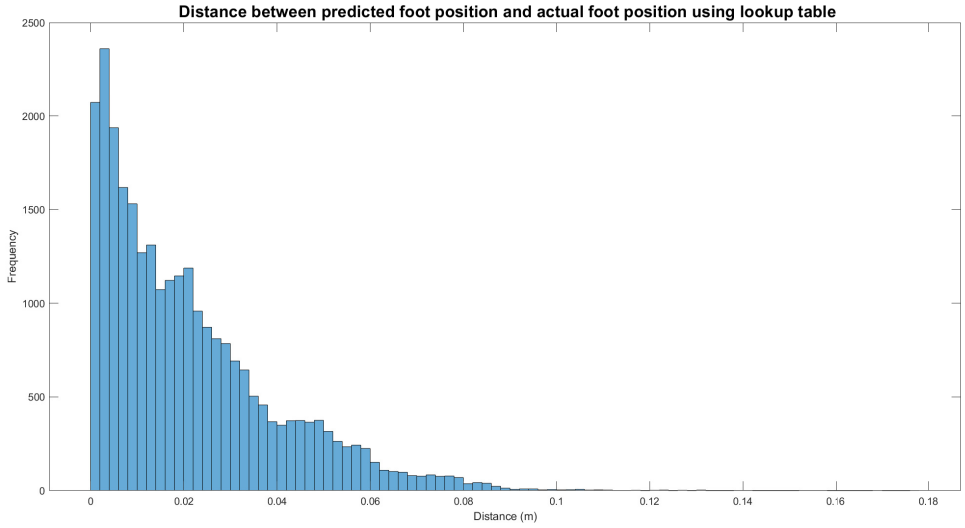


Figure 2-8: Histogram showcasing the error in estimated foot position using a lookup table.

The second filter is also uncomplicated. We check whether or not the lifted leg is the leg we want to lift. This also means that all stances where the robot is standing on all four of its feet are discarded

After these filters have been passed, we must see if the projection of the center of mass in the world frame lies within the area spanned by the contact feet. If it does, the robot should be balanced, and if it does not, the robot will tip over. So the third filter passes through the stances that are balanced and discards the ones that are not.

Referencing equations 2-3, 2-4 and 2-8, the position of the center of mass, com_{ground}, expressed

in the ground frame can be found as follows:

$$\text{com}_{ground} = \frac{\sum_{i=1}^{4} R_{bg}\left(\left(d_i + R_p(\theta_{fh_i})R_r(\theta_{ah_i})\begin{bmatrix}0\\0\\-\frac{1}{2}l_{thigh}\end{bmatrix}\right)m_{thigh}\right)}{4m_{thigh}+4m_{shin}+m_{body}} + \frac{\sum_{i=1}^{4} R_{bg}\left(\left(P_{k_i} + R_r(\theta_{ah_i})R_p(\theta_{fk_i}+\theta_{fh_i})\begin{bmatrix}0\\0\\-\frac{1}{2}l_{shin}\end{bmatrix}\right)m_{shin}\right)}{4m_{thigh}+4m_{shin}+m_{body}} \tag{2-10}$$

Where m_{thigh}, m_{shin}, and m_{body} are the mass of the thigh, shin, and body respectively. Note that we do not include the position and mass of the body in the denominator of equation 2-10, as its position coincides with the origin of the body frame. Furthermore, we can multiply vectors $\begin{bmatrix}0 & 0 & l_{shin}\end{bmatrix}^T$ and $\begin{bmatrix}0 & 0 & l_{shin}\end{bmatrix}^T$ by $\frac{1}{2}$, because we assume that all body parts have equal density throughout.

The fourth filter observes if the pushing limb is configured in a way that allows for a push to be made. If a stance is generated with the pushing limb retracted underneath the body, then it is not useful. We calculate the position of the pushing limb in the world frame and check if its x-coordinate is located fully in front of the body. Furthermore, we check if the pushing limb's foot reaches past its knee in the x-direction and if the z-coordinate of the foot is not higher than the height of the object we wish to push.

The fifth filter prevents that any non-pushing limb might clip into the object that is to be pushed. If the foot closest to the pushing limb reaches further than the pushing foot in the x-direction of the world frame and its y-coordinate in the world frame is located within the range of y-coordinates that the object occupies, the stance is removed from the set of eligible stances.

The inverse kinematics can assess which stances may come close to our intended target, and which may not, which we utilize for the sixth filter. The goal location is known, and defined as the location we want our pushing limb to attain expressed in the world frame. This has to do with how we choose to push the object, a topic discussed in Chapter 3. The z-coordinate is of no importance as long as contact with the object is maintained and the pushing limb remains above the ground. We create a vector containing ten evenly spaced values for the z-coordinate and create ten goals with the same x- and y-coordinate and a varying z-coordinate. Then we use the method described in Section 2-2-2 to make an estimate of how close the pushing limb can get to any of these goals. If the closest it can get to one of these goals is within a threshold of 5.0 cm, then we keep the stance in consideration, otherwise it is discarded.

In order to make the overall control scheme more robust, we must also consider the balance of the robot as it pushes and whether or not any moments created by the push can jeopardize the safety of the robot. Thus, we apply two final filters, filters seven and eight, to make the stance more robust.

The penultimate filter takes an estimate of the final position of the pushing limb, obtained during the filter before, and, using equation 2-10, checks if the position is balanced, similarly to the third filter we described.

Finally for the eighth filter, we take into consideration if the force needed to push the object creates enough moment to make the robot tip over. We will discuss the modeling of our push in detail in Chapter 3, but important information for this filter is that we assume Coulomb friction and a quasi-static push where $F_f = \mu N$, with F_f as the friction force, μ as the coefficient of friction and FN as the normal force. As a consequence of our assumptions the friction force has to be the same throughout the pushing motion and is in the opposite direction of the pushing force. Though we do not know beforehand how our push will look, as that requires planning, we assume that the direction of the pushing force is the vector from the starting position to the estimated goal.

The robot can tip from the moment around the three axes the contact feet span, as depicted in Figure 2-9. We create three different coordinate systems in a similar fashion as described in Section 2-2-1, but with the x-axis coinciding with the relevant axis spanned by two of the contact feet. We then compute the moment around these three x-axes by looking at two forces, the friction force estimated as described above and the gravity from the center of mass. If the moment caused by gravity around all axes is bigger than the moment caused by the friction force, or the moment caused by the friction force has the same sign as the moment caused by gravity, then the stance is preserved.

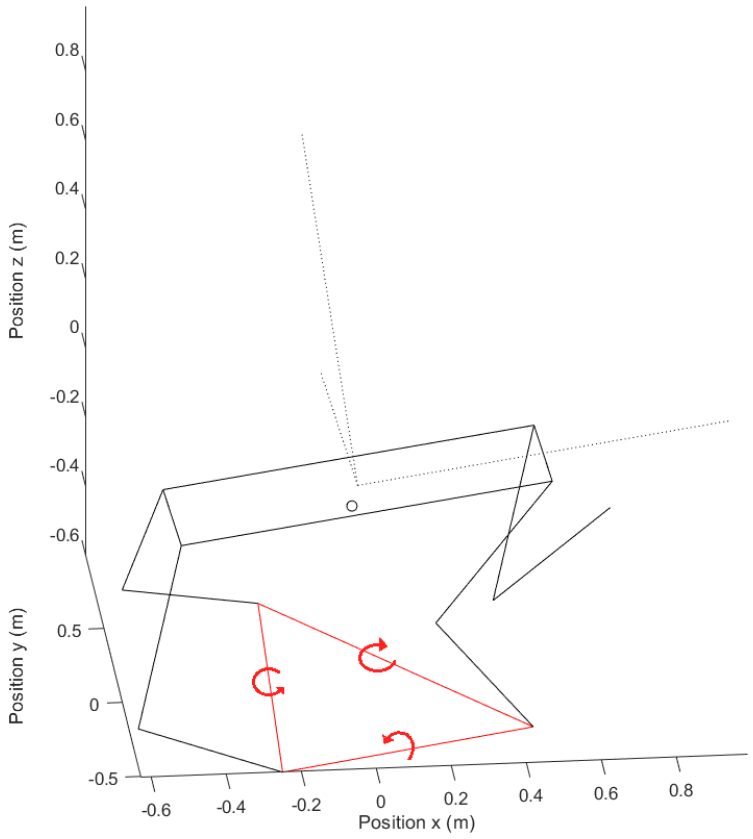


Figure 2-9: Overview of the three axes around which the robot may tilt during pushing.

The filters that employ the inverse kinematics provide no guarantee that the robot will remain balanced because their estimations are speculative, but they should provide some manner of

robustness. We will look back on the balance of the robot in Chapter 3.

When all filters are exhausted, we are left with some number of suitable stances. The stance that is ultimately chosen is the one that has the least displacement of the joint angles compared to the resting stance. This is our measure of optimality. The reasoning behind this measure for optimality is that the stances that alter the joint angles the least, are the ones that are the easiest to achieve in practice. We do not concern ourselves with the locomotion or positioning of the robot, something we discuss further in Chapter 4, but it is in our best interest that the option chosen can ultimately be used in real life scenarios and using the displacement of the joint angles for this seems like a reasonable optimality criterion.

The entirety of the filtering algorithm can be seen in algorithm 1. Here KneesAboveFeet is the filter that checks if all standing feet have a lower z-coordinate in the world frame than their respective knees. LiftedLeg refers to the procedure where we check if the knees are below all contact feet and the appropriate leg is lifted. RestBalance is checks if the projection of the center of mass in the world frame, is located inside of the area spanned by the standing feet. FootConfiguration examines if the pushing limb is configured in a manner that can be conducive to a push. NoClip is the filter that checks whether the other front foot clips into the object that is to be pushed. EstGoalDistance uses the ANFIS networks described in Section 2-2-2 to see if the stance can get within 5.0 cm of the intended goal. Finally, ExtendedBalance and MomentBalance check if the robot is expected to tip over at due to the shifted location of the center of mass and the moment caused by the pushing motion, respectively. When any of the mentioned functions, other than EstGoalDistance, returns 1 that means the filter is passed, otherwise it is failed and the position is considered unfit.

Algorithm 1 Stance filtering algorithm

Input:
12 joint angles: θ_{ah_i}, θ_{fh_i}, θ_{fk_i} for $i = 1, 2, 3, 4$
if KneesAboveFeet $= 1 \wedge$ LiftedLeg $= 1 \wedge$ RestBalance $= 1 \wedge$ FeetConfiguration $= 1 \wedge$ NoClip $= 1 \wedge$ EstGoalDistance ≤ 5 cm $\wedge$ ExtendedBalance $= 1 \wedge$ MomentBalance $= 1$ **then**
Stance is viable
else
Stance is disregarded
end if

Now that algorithm 1 has been established, we have the ability to test whether a generated stance is a suitable candidate for the pushing task, or not. This does not, however, equip us with the knowledge of how we should go about generating these stances. There are several ways to go about this task.

The first and most simple way of generating stances is to do so by creating twelve random joint angles from a uniform distribution over the entire range of motion of each joint. The problems with this approach are obvious: it generates a large number of useless stances and it is computationally expensive. From experience we found that even a number as high as 100000 generated stances does not necessarily lead to one useful one given a certain goal or object properties. If the stance selection algorithm is to be used in real world application, stance generation must be done more efficiently and reliably.

A more sensible approach than simply brute forcing a stance would be to limit the range of motion of each joint in ways that fit the situation at hand, and uniformly distribute joint positions over the limit range of motion per joint. The problem with this approach is that it can be hard to evaluate what joint angles are more reasonable than others. All joint angles are defined from the body frame, and we do not know beforehand how each angle will leave the robot oriented. For this reason we take a different approach.

We generate stances by taking a hand-selected base stance that is appropriate to the pushing goal and the object's properties and take a normal distribution over each angle. This allows us to generate a pose quickly, as more of the candidates are viable and we can keep our sample size much lower.

Another benefit to this approach is that the filters present in algorithm 1 are not entirely comprehensive and brute forcing sometimes leads to a stance that is not usable in the real world, and this approach circumvents that problem. Brute forcing might lead to a stance that passes all the filters, but has the two hind legs crossed, as we do not filter for that. By selecting a base stance first and deviating from it with a normal distribution, we have found that we always come to a usable stance.

Our chosen approach does come with the issue that we need to have a stance that suits the situation a priori and then produce a final stance from that initial base stance. In the context of this thesis, we attain this hand-selected stance by brute forcing an one and checking manually if it is appropriate. This is clearly not the approach that should be taken in real-world applications. Rather, we recommend being able to access such a reference stance quickly, before taking a normal distribution over the joint angles. This can be done, for example, by having an in-memory database with reference stances belonging to an area of pushing goals and object properties or by training a model through supervised learning that can quickly generate such a position based on the pushing goal and object properties. We will revisit the topic of stance generation in Chapter 3, where we discuss its influence on the subsequent push.

2-3 Conclusion

To come to promising stances, we employ algorithm 1. This algorithm takes a generated stance and assesses its viability by running it through a number of filters. These filters rely on the forward kinematics to infer the stance of the robot and the inverse kinematics to estimate its final position.

Solving the inverse kinematics is a more difficult problem than the forward kinematics, and we solve it using ANFIS networks in conjunction with a lookup table. This network makes three separate estimates for the three joint angles of the pushing limbs taking the coordinates of the goal and an estimate of the angle at the knee and as inputs. We show that this approach to solving the inverse kinematics works on a time scale that is appropriate for the use case and with an accuracy that should suffice for filtering.

We then go over the different filters we employ in algorithm 1. Through these filters we run a generated position and we assess whether the stance is geometrically sensible, meaning the correct leg is lifted and all standing feet are below their respective knees, is not configured in a way where the other front leg interferes and can reach the goal. Furthermore, we employ filters that check for balance at the start and end of the robot's motion, to make for a more robust

push. These filters check whether the center of mass is supported by the three standing legs and if the moment caused by the push might imbalance the robot. The final selection, then, is made by choosing the stance from the remaining selection that has the smallest displacement of all joint angles, compared to the resting stance.

The filters that each stance goes through are not so comprehensive that every stance that would be impractical in the real world is filtered out. We can circumvent the impractical stances that do not get filtered out, by first generating a sensible stance and subsequently generating a set of stances based on that first stance, selecting the optimal one from that set.

We are ultimately left with a method that can generate a stance that holds up against a series of filters and was deemed most suitable out of all stances that passed those filters. An appropriate stance is a necessary precondition to a successful push, which is what will be discussed in Chapter 3.

Chapter 3 - Modeling and Control of Pushing

Chapter 2 discussed the problem of stance selection and has left us with a method to come to a stance that makes for a viable push, but how we come to plan this push is still an open-ended question. In this Chapter, we concern ourselves with pushing the object that is obstructing the robot's path to its intended goal. The mechanics of pushing is a subject that has garnered much academic interest, though it has never been implemented on a quadrupedal robot as we present in this thesis. We will explore the mechanics of pushing and see how we can incorporate the mechanics into our control framework, which utilizes Model Predictive Control (MPC). This will leave us with a proposed method of pushing away objects during missions.

In broad terms, our methodology to come to a successful model predictive controller relies on us constraining the motion of the end-effector in such a way that it and the object form a sticking contact. If this is achieved, the motion of the end-effector and object are identical. These constraints depend both on certain physical properties of the object and the capabilities of the robot itself. We will also see that the robot's stance has influence on the pushing motion.

In this Chapter we first address the mechanics of pushing, explaining how we model a push and discuss how we might implement the model into our control framework. Then we discuss the experiment we conducted, which serves to investigate whether our model and the assumptions it is based on are appropriate and applicable to our use case. Finally, we elaborate on the implementation of MPC and its results in simulation. Section 3-1 discusses the mechanics of pushing. In Section 3-2 we delve into an experiment we conducted and compare it, as well as other experimental findings, to the theory we want to utilize for our MPC. In Section 3-3 we discuss the implementation of MPC and showcase its performance. Finally, we summarize this Chapter and present our conclusions on modeling and control in Section 3-4.

3-1 Mechanics of Pushing

The act of pushing is heavily researched in the field of robotics as the act of moving an object from one place to another is essential in a great many applications and pushing is one of the most common methods of moving objects. As it is such a relevant topic, there is an exhaustive amount of literature on the modeling of pushing. Stüber, Zito and Stolkin identify six different categories for modeling pushing: analytical models, hybrid models, physics engines, data driven models, and deep learning models [8].

In this thesis we work with an analytical method, as it is more easily reproduced and integrated into an MPC framework, as we will see in Section 3-3. We make use of motion cones, a concept first introduced by Mason [13], which represent the set of motions an object can make when undergoing a frictional push. If the motion of the pusher stays within the motion cone, the contact between the pusher and the object is a sticking contact, meaning that the two do not move relative to one another. We will use said motion cones to restrict the motion of our end-effector in such a way that the contact between the object and end-effector is always a sticking contact. We will now go over how the motion cone is constructed, by first looking at two concepts that are imperative to formulating what a motion cone is. The first topic we will go over is the limit surface, then we discuss the generalized friction cone, and from those two we create motion cones. When we create motion cones, we follow the approach of Chavan-Dafle, Holladay and Rodriguez [14, 15]. It is their methodology that we will describe in the following Sections.

3-1-1 The Limit Surface

The friction limit surface describes all possible friction forces that can occur when an object slides across a support surface [16]. If an object is pushed by a friction wrench $w = \begin{bmatrix} f_x & f_y & m_z \end{bmatrix}^T$ that is contained within in the limit surface, where f_x and f_y denote the forces in the x- and y-direction and m_z denotes the moment around the z-axis, then the object remains stationary. If the object is affected by a friction wrench that lies on the limit surface, however, it moves quasi-statically with its twist $t = \begin{bmatrix} v_x & v_y & \omega_z \end{bmatrix}^T$ normal to the limit surface at the applied wrench w, where v_x and v_y are the velocities in the x- and y-direction and ω_z is the rotation speed about the z-axis [5]. The quasi-static assumption we work with is often employed in analytical models of pushing [8] and presupposes that all motion is instantaneous, so dynamics are neglected. This assumption is appropriate for applications where velocities are relatively low.

Figure 3-1(a) gives a depiction of the limit surface and a composite wrench cone, or generalized friction cone, that we will learn to construct in the next Section. At the intersection of any of the vectors of the wrench cone with the limit surface, we can see the associated twist vector that is normal to the limit surface at that intersection.

The limit surface can be approximated by an ellipsoid [17, 18], which is commonplace in literature as it makes for efficient simulation and planning [19, 20]. If $w = \begin{bmatrix} f_x & f_y & m_z \end{bmatrix}^T$ is the frictional wrench on an object, then the ellipsoidal approximation of the limit surface can be represented as follows:

$$w^T A w = 1 \tag{3-1}$$

with:

$$A = \begin{bmatrix} \frac{1}{(\mu_c N)^2} & 0 & 0 \\ 0 & \frac{1}{(\mu_c N)^2} & 0 \\ 0 & 0 & \frac{1}{(k\mu_c N)^2} \end{bmatrix}$$

Where μ_c is the coefficient of friction between the object and the support surface, N is the normal force, and k is an integration constant. When we assume uniform pressure distribution of the object's weight onto the support surface, we can say that the integration constant $k \approx 0.6r$, with r being the radius of the contact [21].

The ellipsoidal representation of the limit surface necessitates that the direction of the friction forces and the velocities in the x- and y-direction are parallel and opposite [19]. This is in concordance with the method we use to calculate moments in Section 2-2-3. If the twist at the contact v_s are known, we can find the friction wrench w_s as follows [14]:

$$w_s = \frac{A^{-1} v_s}{\sqrt{v_s^T A^{-1} v_s}} = \frac{\mu_c N B^{-1} v_s}{\sqrt{v_s^T B^{-1} v_s}} = \mu N \bar{w}_c \tag{3-2}$$

with:

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{k^2} \end{bmatrix}$$

Here $\bar{w_s}$ the normalized friction wrench.

Conversely, if we know the friction wrench w_s we can solve for v_s as follows [15]:

$$v_s = \bar{k} B \bar{w_c} \tag{3-3}$$

Where $\bar{k}$ is a scalar greater or equal to zero which scales the twist at the contact v_s. We will see that equation 3-3 can be used to construct the motion cone.

3-1-2 Generalized Friction Cone

The limit surface helps us model the interaction between the pushed object and the support surface underneath it. We use the concept of generalized friction cones, also called composite wrench cones, to model the interaction between the pusher and the pushed object. Generalized friction cones, introduced by Erdmann [22], models the permissible pushing forces of a line contact by combining multiple Coulomb friction cones. To fully understand this procedure, we must address the concept of stable pushing and we will show an example of how to construct a generalized friction cone.

The concept of stable pushing [13, 23] relies on line contacts, that is to say contacts that make several point contacts with an object, making certain restricted movements that ensure that the pusher and the pushed object stick to one another. It should be noted that this approach models the object that is to be pushed as two dimensional, viewed from the top down. We rely on the principle of stable pushing in this thesis, requiring that we make a line contact with the pushed object. The generalized friction cone takes the Coulomb friction cones at each end of the line contact and forms a convex hull of these two friction cones [15].

A friction cone for a line contact in two dimensions can be constructed if we know the coefficient of friction μ_p between the pusher and the pushed object and the location of the point contact. In Figure 3-1(b) we see an object and a pusher that makes a line contact with the object. At the extremities of the line contact two vectors are drawn: w_1 and w_2 on the left extremity and w_3 and w_4 on the right extremity. The angle α between the y-axis and the vectors is computed as $\alpha = \pm \arctan \mu_p$ [19, 5]. For each vector the moment around the center of mass (which is assumed to coincide with the center of friction) is calculated. We can then construct the generalized friction cone W_{pusher} as depicted in Figure 3-1(c). The generalized friction cone W_{pusher} can be characterized as:

$$W_{pusher} = \{w_{pusher} = f_p | f_p \in \mathrm{FC}_{pusher}\} \tag{3-4}$$

Where f_p are the contact forces and FC_{pusher} is the friction cone at the two pusher point contacts on each end of the line contact.

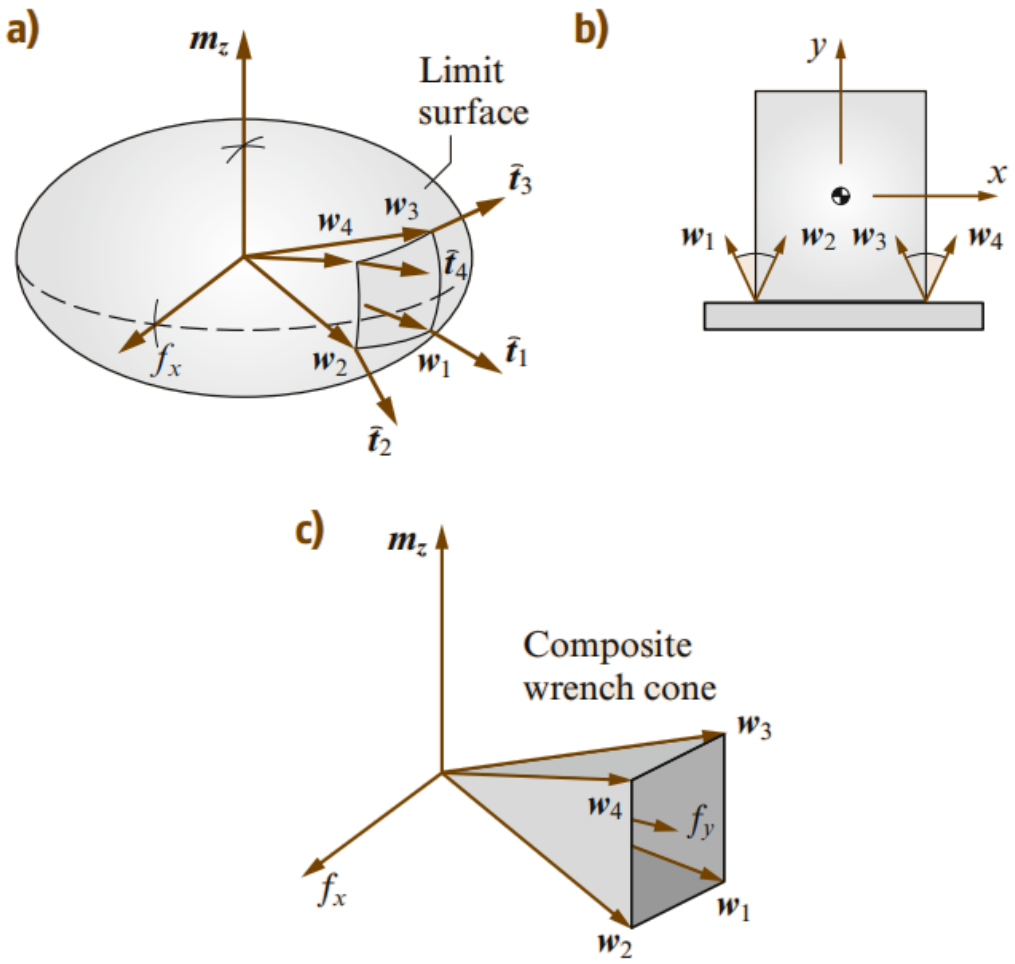


Figure 3-1: (a) The limit surface and a wrench cone that intersects it. We can map the twists at the intersection of the composite wrench cone. (b) Contact between a pushed object and a pusher. We can calculate the generalized friction cone from this contact. (c) A composite wrench cone, also called a generalized friction cone. Adapted from [5].

3-1-3 Motion Cones for Flat, Level Surfaces

The limit surface and generalized friction cones give us the tools to create motion cones. Under the quasi-static assumption, when pushing on a flat surface the only relevant forces are the pushing force and the friction force, which are equal in magnitude and opposite in direction. We can simply state:

$$-\bar{w}_s = \bar{w}_{pusher}, \quad w_{pusher} \in W_{pusher} \tag{3-5}$$

For cases where we are not working with a flat, level surface, gravity cannot be neglected and the expression in equation 3-5 becomes more complicated. As these cases are not relevant to our use case, we will not discuss them, but a detailed explanation can be found in [14] and [15].

The set of all possible support contact wrenches W_s is the negative of the set of all possible pusher wrenches W_{pusher}, meaning $W_s = -W_{pusher}$. That implies that if we know the properties of the object we are pushing and we know where the pusher is located on the object, we can construct W_s. To find the motion cone from W_s, we have to map it through equation 3-3. This does leave us with an unknown variable $\bar{k}$, which scales the motion cone. This is not a big problem, however, as we will see in Section 3-2 that, if our velocities are low enough, the motion cones accurately describe when sticking contact and when slipping contact takes place.

3-2 Experimental Validation of Motion Cones

In Section 3-1 we have devised a method to construct motion cones, which define the velocities that a pusher may apply to an object and have the contact stick. Before we implement motion cones into the controlled push, we want to test their legitimacy through experimentation. We will also mirror our results to findings in literature to further justify our usage of motion cones as an integral part of our approach to a controlled push.

Figure 3-2 displays the experimental setup used to administer pushes to test the slip and stick conditions that motion cones suggest. The robot arm used to push the box depicted in Figure 3-2 is the KUKA LBR IIWA 7 R800, operated through ROS 1. The cardboard box that it pushes is weighed down by a weight of 1.0 kg at its center and pushed by a pusher that is attached to the robot arm. The dimensions of the box are 33.4 cm in length, 23.7 cm in width and 20.3 cm in height. The pusher is 10.0 cm in both width and height. The mass of the box, weighed down by the weight inside of it, is 1.19 kg. The coefficient of friction between the box and the table supporting it was measured to be 0.26, while the coefficient of friction between the box and the pusher was measured to be 0.25.

Figure 3-2: The experimental setup we employed to test the validity of motion cones.

For each push, the pusher and the box make contact at their respective centers, and the robot arm is instructed to move its end-effector a short, randomized distance in the x- and y-direction. The speed of the end-effector is measured and sticking or slipping contact is observed.

The pusher is free to swivel during pushes, meaning that the box does not rotate during the administered pushes. This limits us to only a part of the motion cone, namely the part in which the object does not rotate. We choose to do this, because in our use case we will also not be able to induce rotation with the end-effector, as some sort of wrist joint would have to exist in order to do that. Figure 3-3 gives a representation of the twists that fall into this category, the red lines signify the boundaries of the motion cone where the rotation speed $\omega = 0$ rad/s, which are the twists that we will test.

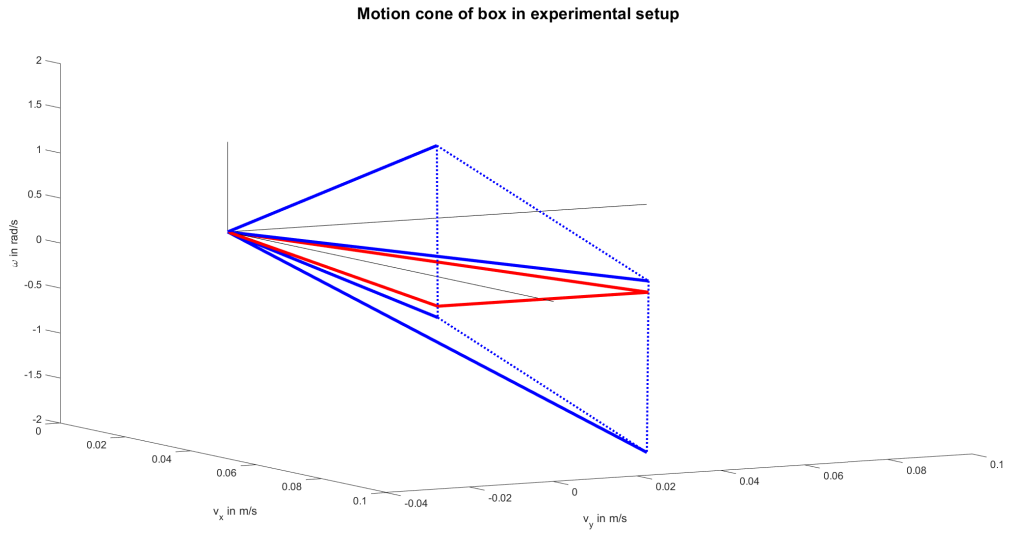


Figure 3-3: The motion cone for the cardboard box in the experimental setup. The red lines within the motion cone mark the part of the motion cone where rotational speed is 0 rad/s.

In total, we performed 200 measurements, observing sticking contact 109 times and slipping contact 91 times. The observations are shown in Figure 3-4, which plots the velocities of the pushes and displays whether or not they fall within the part of the motion cone that was displayed in red in Figure 3-3. All measurements that lie within the bounds of the motion cone are expected to represent a sticking contact and all those outside the motion cone a slipping contact. What we observed is that of the 115 measurements within the motion cone 94 were observed to stick and 21 to slip and of the 84 measurements outside of the motion cone 70 slipped and 15 stuck. This means that the prediction accuracy for sticking contact is 81.7% and the prediction accuracy for slipping contact is 82.4%.

These results are encouraging, as they compare similarly to other results where motion cones were successfully implemented into a control method. Chavan-Dafle, Holladay and Rodriguez [14, 15] performed thousands of measurements each for four different configurations of a gripped object being pushed at one face by a pusher at different grasp forces. The predictive ability of the motion cones constructed for this set of configuration was similar to ours with most predictions being $\geq 80\%$ accurate. In their work, the implementation of motion cones

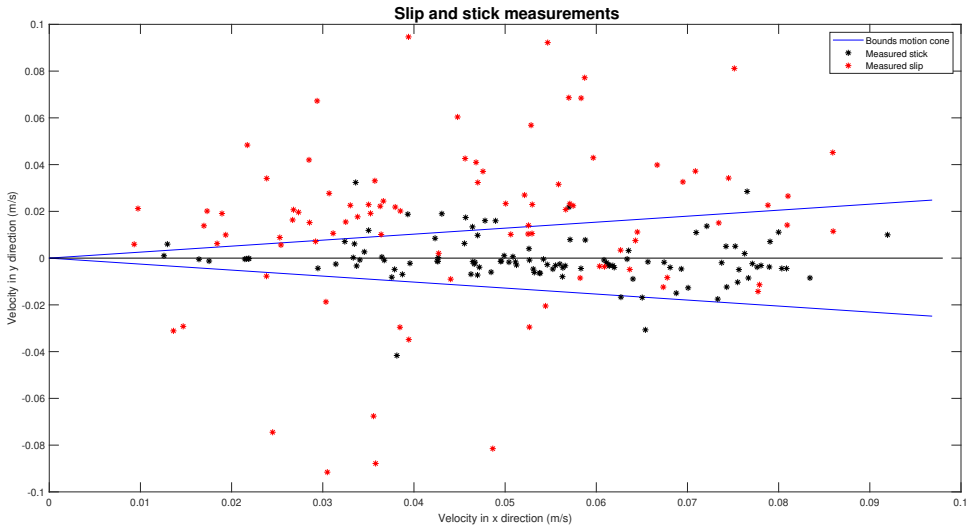


Figure 3-4: Experimental validation of the motion cone.

allows for dexterous manipulations of objects that are gripped between two simple robotic fingers. The manipulations achieved through their methodology are more complicated and require higher precision than the pushing will reasonably need in our use case.

Hogan and Rodriguez [24] also make successful use of motion cones in their paper. The problem they are tasked with is the pushing of an object with a point contact making use of hybrid MPC and using motion cones to distinguish between the different modes. Though their paper provides no experimental validation of motion cones, they are integral to their controller.

Although the motion cone we constructed lines up similarly in performance to those calculated and verified in [14, 15], we are still left with the issue that we do not know at what velocity the quasi-static assumption becomes too inaccurate to rely on. In equation 3-3, the variable $\bar{k}$ is still an unknown scalar that scales the velocity. We argue, however, that as long as velocities remain low enough, we can make use of motion cones when designing our controller. As can be seen in Figure 3-4, the velocity in the x-direction never exceeds 0.1 m/s, which we deem an appropriate velocity for our use case.

3-3 Control

We have opted to employ MPC in our use case, as it is a well-known, well-studied control method that has applications in many fields, including that of non-prehensile manipulation, and it is suitable for the implementation of motion cones as constraints. MPC is a control method where a cost function over a finite horizon is optimized at each time step, using a model to predict how the optimized inputs affect the cost function over the finite time horizon.

In order to create an MPC controller appropriate for our use case, we must first define the exact problem we are trying to solve and the assumptions we are building on. Furthermore, we need to specify our model, our constraints and the goal function. When all the ground work has been established we can evaluate the performance of our controller at different goals

and different stances, looking at its performance and whether or not it jeopardizes the balance of the robot.

3-3-1 Problem and Assumptions

The problem we intend to solve with our MPC controller is solely that of moving our end-effector to the intended goal, whilst constraining the motion in a fashion that keeps its movement within the motion cone. By keeping the twist of the end-effector constrained within the motion cone, more specifically the part of the motion cone where the rotation speed is 0 rad/s, we ensure that there is no relative movement between the pusher and the pushed object. This allows us to formulate the cost function in terms of the position of the end-effector, rather than that of the object.

Defining the problem in the way that we did, presupposes that a number of assumptions are met. Besides the assumptions mentioned in Section 2-1, we make the following, additional assumptions:

- The robot can take on the stance generated by the procedure described in Chapter 2.
- The raised limb makes contact with the object.
- The pushed object is a cuboid.
- We can directly control each joint angle of the robot.
- The pressure that the pushed object exerts on the ground is evenly distributed across its area.
- The limbs that make contact with the ground are static. Their angles do not change as the robot pushes away the object.

One more important subject we need to address before we move onto the topic of control, is that of performing a stable push. All quadrupedal robots available have have feet that can most reasonably be described as point contacts. If we want to employ stable pushes by constraining our movement in accordance with the appropriate motion cones, then we must establish a line contact with the pushed object. We will approach the push as if there is an attachment at the robot's end-effector that is 10.0 cm by 10.0 cm. This attachment does not function as a wrist, meaning it cannot induce rotation of the pushed object.

Now that we have established the problem we want to solve and all important assumptions have been addressed, we can look into designing a controller that can move the pushed object to its intended target.

3-3-2 MPC

We will first discuss the model we use for the MPC controller. As we do not directly concern ourselves with the position of the object and the dynamics of pushing are abstracted away using the quasi-static assumption and constraining our movement with motion cones, we can take the position of the arm as the basis for our model.

We use equation 2-4 to calculate the position of the foot at each time step and the rotation matrix R_{bg} from equation 2-8 to transform the position of the foot to the world frame.

In our MPC problem, we optimize over our joint angles and are tasked with the goal of getting as close to a known target destination as possible. Let us first establish the decision variables of the optimization problem, which are captured with vector θ :

$$\theta = \begin{bmatrix} \theta_{ah_i}(dt) & \theta_{fh_i}(dt) & \theta_{fk_i}(dt) \dots \theta_{ah_i}(N \cdot dt) & \theta_{fh_i}(N \cdot dt) & \theta_{fk_i}(N \cdot dt) \end{bmatrix}^T \tag{3-6}$$

Note that the vector of decision variables θ is $3N$ long, where N is the number of time steps we opt for in the optimization problem.

With our decision variables established, we can look to the cost function, which should be higher when we the end-effector is further away from its intended goal. We argue that the z-coordinate of this goal is unimportant, as long as the z-coordinate of the end-effector is not located above the object or below the standing feet, which we can enforce through our constraints. An intuitive choice for a cost function, then, is one that penalizes being far away from the goal in the x- and y-direction in the world frame. If we call the x-coordinate of the foot and goal at time t $f_x(t)$ and g_x respectively, the y-coordinate of the foot and goal $f_y(t)$ and g_y respectively and we call the number of time steps of the finite horizon N with time step dt, then we define our cost function as:

$$C = \sum_{i=1}^{N} \sqrt{(f_x(i \cdot dt) - g_x)^2 + (f_y(i \cdot dt) - g_y)^2)} \tag{3-7}$$

There are a number of constraints, both linear and nonlinear, that we want to impose on our optimization problem. As for the linear constraints, the joint angles cannot change too rapidly and the joint angles may never move beyond their range of motion. We address that latter concern by constraining the upper and lower bound of θ as follows:

$$\begin{bmatrix} \theta_{ah_{min}} \\ \theta_{fh_{min}} \\ \theta_{fk_{min}} \\ \vdots \\ \theta_{ah_{min}} \\ \theta_{fh_{min}} \\ \theta_{fk_{min}} \end{bmatrix} \leq \theta \leq \begin{bmatrix} \theta_{ah_{max}} \\ \theta_{fh_{max}} \\ \theta_{fk_{max}} \\ \vdots \\ \theta_{ah_{max}} \\ \theta_{fh_{max}} \\ \theta_{fk_{max}} \end{bmatrix} \tag{3-8}$$

Where $\theta_{ah_{max}}$, $\theta_{fh_{max}}$, and $\theta_{fk_{max}}$ denote the maximum joint angles for adduction/abduction at the hip, flexion/extension at the hip and flexion/extension at the knee, respectively, and $\theta_{ah_{min}}$, $\theta_{fh_{min}}$, and $\theta_{fk_{min}}$ denote the minimum joint angle for said angles.

We can constrain the rate of change of the angles as follows:

$$-\begin{bmatrix} dt \cdot \omega_{ah_{max}} \\ dt \cdot \omega_{fh_{max}} \\ dt \cdot \omega_{fk_{max}} \\ \vdots \\ dt \cdot \omega_{fk_{max}} \end{bmatrix} \leq \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \ddots & & \vdots & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \theta \leq \begin{bmatrix} dt \cdot \omega_{ah_{max}} \\ dt \cdot \omega_{fh_{max}} \\ dt \cdot \omega_{fk_{max}} \\ \vdots \\ dt \cdot \omega_{fk_{max}} \end{bmatrix} \tag{3-9}$$

Where $\omega_{ah_{max}}$, $\omega_{fh_{max}}$, and $\omega_{fk_{max}}$ are the maximum angular velocity for adduction/abduction at the hip joint, flexion/extension at the hip joint, and flexion/extension at the knee joint respectively.

We must also ensure that the rate of change between the initial condition and the first time step after the initial condition are constrained. This constraint is linear, but we have to account for the fact that the value of the initial condition changes at every instance we solve the optimization problem. This constraint can be written as:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 & 0 \end{bmatrix} \theta \leq \begin{bmatrix} dt \cdot \omega_{ah_{max}} + \theta_{ah_1}(0) \\ dt \cdot \omega_{fh_{max}} + \theta_{fh_1}(0) \\ dt \cdot \omega_{fk_{max}} + \theta_{fk_1}(0) \\ dt \cdot \omega_{ah_{max}} - \theta_{ah_1}(0) \\ dt \cdot \omega_{fh_{max}} - \theta_{fh_1}(0) \\ dt \cdot \omega_{fk_{max}} - \theta_{fk_1}(0) \end{bmatrix} \tag{3-10}$$

Where $\theta_{ah_1}(0)$, $\theta_{fh_1}(0)$, $\theta_{fk_1}(0)$ are the current joint angles.

As mentioned, we also have a number of nonlinear constraints that we have to factor into the optimization problem. We must make sure that the end-effector stays above the standing feet and below the top of the box, and we need to ensure that the end-effector's twist is located within the motion cone. We enforce that the end-effector does not exceed these lower and upper bounds as follows:

$$f_{stand_z} + \epsilon \leq \begin{bmatrix} f_{i_z}(dt;\theta) \\ f_{i_z}(2 \cdot dt;\theta) \\ \vdots \\ f_{i_z}(N \cdot dt;\theta) \end{bmatrix} \leq h_{object} - \epsilon \tag{3-11}$$

Where f_{stand_z} and f_{1_z} are the z-coordinate of the standing feet and the z-coordinate of the raised foot (foot i) respectively, h_{object} is the height of the object and ϵ is a margin to ensure that the end-effector does not operate closer to the bottom or top of the object than is deemed desirable. The z-coordinates of the standing feet and the raised foot are calculated using the forward kinematics discussed in Section 2-2-1.

We can enforce the constraints on the velocity of the push, by first setting a limit to how fast the end-effector can move in the x-direction and then constraining the velocity in the y-direction as a function of the velocity in the y-direction, based on the motion cone. First we constrain the velocity in the x-direction, v_{i_x}:

$$0 \leq \begin{bmatrix} v_{i_x}(dt;\theta) \\ v_{i_x}(2 \cdot dt;\theta) \\ \vdots \\ v_{i_x}(N \cdot dt;\theta) \end{bmatrix} \leq v_{x_{max}} \tag{3-12}$$

Where $v_{x_{max}}$ denotes the maximum velocity in the x-direction. We cannot set a definitive value for $v_{x_{max}}$ on a case per case basis, but our findings from Section 3-2 suggest that a value around 0.08 m/s might be an appropriate for most scenarios.

Then we can constrain the velocity in the y-direction, v_{i_y}, by relating it to velocity v_{i_x}. From Figures 3-3 and 3-4 we can tell that the section of the motion cone that we operate in, can be defined by two lines that make a certain angle with the x-axis. We will refer to these angles as α_{high} and α_{low}, where α_{high} is the larger angle and α_{low} is the smaller angle with respect to the x-axis. With these definitions clear, we can formulate the nonlinear constraint on v_{i_y}:

$$\begin{bmatrix} v_{i_x}(dt;\theta)tan(\alpha_{low}) \\ v_{i_x}(2\cdot dt;\theta)tan(\alpha_{low}) \\ \vdots \\ v_{i_x}(N\cdot dt;\theta)tan(\alpha_{low}) \end{bmatrix} \leq \begin{bmatrix} v_{i_y}(dt;\theta) \\ v_{i_y}(2\cdot dt;\theta) \\ \vdots \\ v_{i_y}(N\cdot dt;\theta) \end{bmatrix} \leq \begin{bmatrix} v_{i_x}(dt;\theta)tan(\alpha_{high}) \\ v_{i_x}(2\cdot dt;\theta)tan(\alpha_{high}) \\ \vdots \\ v_{i_x}(N\cdot dt;\theta)tan(\alpha_{high}) \end{bmatrix} \tag{3-13}$$

With that, we have addressed everything needed to give get a full understanding of how we structure our MPC optimization problem. Equation 3-14 shows the optimization problem concisely in its entirety.

$$\begin{bmatrix} f_x(t) \\ f_y(t) \\ f_z(t) \end{bmatrix} = R_{bg}\left(d_i + R_p(\theta_{fh_i}(t))R_r(\theta_{ah_i}(t)) \begin{bmatrix} 0 \\ 0 \\ -l_{thigh} \end{bmatrix} + R_r(\theta_{ah_i}(t))R_p(\theta_{fk_i}(t)+\theta_{fh_i}(t)) \begin{bmatrix} 0 \\ 0 \\ -l_{shin} \end{bmatrix} \right)$$

$$\min_{\theta(dt),\ldots,\theta(N\cdot dt)} \sum_{i=1}^{N} \sqrt{(f_x(i\cdot dt)-g_x)^2 + (f_y(i\cdot dt)-g_y)^2)}$$

$$s.t. \begin{cases} \text{Equation } (3-8) \\ \text{Equation } (3-9) \\ \text{Equation } (3-10) \\ \text{Equation } (3-11) \\ \text{Equation } (3-12) \\ \text{Equation } (3-13) \end{cases} \tag{3-14}$$

3-3-3 MPC in simulation

The optimization problem posed in Equation 3-14 can be solved as a nonlinear optimization problem. We tested its performance in simulation, having the end-effector navigate to three specific goals pushing four specific boxes. The performance of the end-effector in simulation can be seen in Tables 3-1, 3-2, 3-3, and 3-4. In all cases, the end-effector starts in the middle of the box and the algorithm used to solve the optimization problem, which is run in MATLAB, is Sequential Quadratic Programming (SQP). Every push for each of the three goals and four boxes was run 100 times (for a total of 1200 planned pushes), with a finite horizon of 10 time steps and a time step size of 0.25 seconds. The average runtime of simulating a run was approximately 1.8 seconds, whilst the simulated time is 10 seconds. The maximum speed of the push is 0.08 meters per second, as to be consistent with our findings in Section 3-2. We assume the end-effector of the pushing limb is equipped with a 10 cm by 10 cm pusher and all pushes are performed by the front left leg.

The three goals we examined are:

- Goal 1: Goal position is 20 cm removed from the original position of the end-effector in the x-direction and 5 cm in the y-direction. Angle with x-axis $\phi \approx 14.0°$.
- Goal 2: Goal position is 15 cm removed from the original position of the end-effector in the x-direction and 15 cm in the y-direction. Angle with x-axis $\phi = 45°$.
- Goal 3: Goal position is 30 cm removed from the original position of the end-effector in the x-direction and 2 cm in the y-direction. Angle with x-axis $\phi \approx 3.81°$.

The relevant properties of the four boxes we examined are:

- Box 1: 50 cm by 50 cm by 50 cm, $\mu_{pusher} = 0.3$, $\mu_{contact} = 0.5$, $m_{box} = 0.5$ kg. Motion cone angle with pusher at center: $\gamma \approx 11.3°$.
- Box 2: 30 cm by 30 cm by 30 cm, $\mu_{pusher} = 0.3$, $\mu_{contact} = 0.5$, $m_{box} = 0.5$ kg. Motion cone angle with pusher at center: $\gamma \approx 16.7°$.
- Box 3: 50 cm by 50 cm by 50 cm, $\mu_{pusher} = 0.3$, $\mu_{contact} = 0.8$, $m_{box} = 3$ kg. Motion cone angle with pusher at center: $\gamma \approx 11.3°$.
- Box 4: 50 cm by 50 cm by 50 cm, $\mu_{pusher} = 0.1$, $\mu_{contact} = 0.5$, $m_{box} = 0.5$ kg. Motion cone angle with pusher at center: $\gamma \approx 5.71°$.

Important to consider for our results, as depicted in Tables 3-1, 3-2, 3-3, and 3-4, are the angles ϕ and γ, which are the angle the goal makes with the x-axis and the angle the motion cone makes with the x-axis, respectively. Note that in our simulation the section of the motion cone is symmetrical, because the pusher is located at the middle of the face of the box it is pushing against, which means that the section of the motion cone where there is no rotation makes an angle γ and $-\gamma$ with the x-axis. If the pusher is located anywhere else on the face it is pushing against, then the motion cone is not symmetrical.

From the Tables we can tell that if $\gamma > \phi$, the performance of the controller is significantly better than if the reverse is true. The averages for the distance between the end-effector in the x-direction and the y-direction, $\overline{d_x}$ and $\overline{d_y}$, and the average total distance, $\overline{d_{goal}}$, is orders of magnitude smaller if the goal is contained within the motion cone. This is unsurprising, as the constraints on the velocity in the y-direction, given in equation 3-13, enforce that the goals that make a larger angle with the x-axis than the motion cone itself cannot be reached. Instead, the controller pushes the box along the edge of the motion cone, to come as close as possible.

What cannot be directly inferred from the Tables, but is noteworthy, is that the cases where $\gamma < \phi$, the controller only gives a handful of different solution, whilst in the reverse case, each solution was unique. This supports the notion that the controller pushes the box along the edge of the motion cone if $\gamma < \phi$ and finds the same local optima to do so. In the case where $\gamma > \phi$, this is not observed as there does not exist a select number of paths to the goal that are better than all others.

It is worth discussing what factors make a goal easier or more difficult to reach. As for the goal itself, the smaller the angle with the x-axis, the easier it is to reach. If the end-effector

can operate at or close to the center of the motion cone, the controller consistently produces the best results. We can see from Table 3-2 that the controller performs better on goal 3 than on goal 1, even though both goals make a smaller angle with the x-axis than the motion cone does.

As for the properties of the box, it is important to note that the mass and coefficient of friction of the box do not influence the shape of the motion cone in the case when pushing on a flat, level surface [15]. This explains why boxes 1 and 3 have the same angle γ. Three other properties do influence the shape of the motion cone and therefore the range of goals that can be attained successfully: the dimensions of the pusher, the dimensions of the box and the coefficient of friction between the box and the pusher.

The dimensions of the pusher and the dimensions of the box influence the motion cone in opposite ways. A larger (wider) pusher makes for a larger angle γ and a larger box, both in length and in width, make for a smaller angle γ. A larger coefficient of friction between the pusher and the box makes for a larger angle γ, which can be inferred from Tables 3-1 and 3-4, since boxes 1 and 4 only differ in μ_{pusher}. If we were to design a pusher that is to be attached to one of the robot's feet, it would have to be made of a material that has a high coefficient of friction with most materials in order to maximize the range of locations to where we can effectively push objects.

We must also discuss the robustness of the controller. Through the heuristics discussed in Chapter 2, we tried to achieve a stance selection algorithm that allows the pushing stage to be both successful in reaching the goal and robust to tipping over, either because of the moments from the push or from the displacement of the center of mass during the push.

Firstly, we will discuss the controller's robustness against the moments. We calculated the moment around each of the three axes spanned by the standing feet, with the method described in Chapter 2, and if at any time step the moment around one of those axes would cause tipping, we consider that push a failure. From Table 3-3, compared to the other three Tables, we can tell that higher friction forces make for a more difficult push, which is a logical conclusion to make. Where boxes 1, 2, and 4 consistently achieve a success rate r_{moment} of higher than 90%, box 3 performs significantly worse. That being said, typically when failure was observed, it was only during a single time step of the push, accounting for 0.25 seconds. It remains to be seen whether this would cause failure in practice, though we must clearly be more careful when considering pushing heavier objects.

Secondly, we can observe that the displacement of the center of mass is not a problem in more than 90% of the cases across all different boxes and goals, with the success rate r_{com} averaging to around 96% across all simulations. The properties of the box are naturally not influential, as the position of the center of mass is not reliant on the box, and the location of the goal also does not seem to make a difference. Again, it is doubtful if the cases where the robot does not remain balanced would always result in failure, as the contact with the object might keep the robot upright.

Figures 3-5 and 3-6 show a push that is typical of what happens when the angle with the x-axis of the motion cone, γ, is smaller and larger than the angle of the goal with the x-axis, ϕ, respectively. The Figures show a push with box 1 to goals 2 and 3, respectively, where 2 is difficult to attain and 3 easier.

As mentioned, if $\theta < \phi$, performance worsens, which we can see in Figure 3-5. We see that

Box 1	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)	r_{com}	r_{moment}
Goal 1	0.00288	0.0144	0.0147	95%	92%
Goal 2	0.0231	0.116	0.118	95%	92%
Goal 3	$2.63 \cdot 10^{-7}$	$1.90 \cdot 10^{-7}$	$2.81 \cdot 10^{-7}$	94%	91%

Table 3-1: Results simulation for box 1. Motion cone angle $\gamma \approx 11.3°$.

Box 2	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)	r_{com}	r_{moment}
Goal 1	$4.02 \cdot 10^{-4}$	0.00134	0.00140	99%	91%
Goal 2	0.0297	0.0991	0.103	93%	93%
Goal 3	$3.32 \cdot 10^{-7}$	$1.80 \cdot 10^{-7}$	$4.09 \cdot 10^{-7}$	96%	95%

Table 3-2: Results simulation for box 2. Motion cone angle $\gamma \approx 16.7°$.

Box 3	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)	r_{com}	r_{moment}
Goal 1	0.00316	0.0158	0.0450	96%	88%
Goal 2	0.0231	0.116	0.118	98%	85%
Goal 3	$7.53 \cdot 10^{-8}$	$9.36 \cdot 10^{-8}$	$1.35 \cdot 10^{-7}$	94%	77%

Table 3-3: Table containing all simulation values for box 3. Motion cone angle $\gamma \approx 11.3°$.

Box 4	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)	r_{com}	r_{moment}
Goal 1	0.00561	0.0313	0.0332	98%	91%
Goal 2	0.0135	0.134	0.135	92%	92%
Goal 3	$1.50 \cdot 10^{-7}$	$7.24 \cdot 10^{-7}$	$7.79 \cdot 10^{-7}$	98%	93%

Table 3-4: Results simulation for box 4. Motion cone angle $\gamma \approx 5.71°$.

the x-coordinate of the goal is overshot, while the end-effector does not get close to the y-coordinate of the goal. The cost function given by equation 3-7 values the distance in both the x- and y-direction equally and the optimal solution then takes on such a form. If we were to value the distance in the x-direction as more important than the y-direction, or vice versa, the results would reflect that.

In Figure 3-6 we see that if $\phi < \theta$, the controller has no trouble attaining both the x- and y-coordinate of the goal accurately.

Adherence to the constraints set by equations 3-9,3-10, 3-11, 3-12, and 3-13 during the push must also be examined. The only constraint that we observed being broken is the constraint pertaining the rotational speed, given by equation 3-9, in the first time step. Typical behavior for the controller is to move the end-effector along the z-axis to as low, or close to as low, as the constraints given by equation 3-11 allow the end-effector to go. A reason for this action might be that the range of motion is more advantageous at a lower z-coordinate. This movement is often performed by rotating the joint angles at a faster rotation speed than is allowed. The constraint on the joint that flexes the hip is most often the one that is broken, though we have observed the constraint on the joint responsible for the adduction of the hip also being broken. A case where the hip flexing joint rotates faster than is allowed can be seen in Figure 3-7. Whilst this violation of the constraints is unfortunate, it is not one that forms a threat to the quality of the push; the x-coordinate and y-coordinate barely change

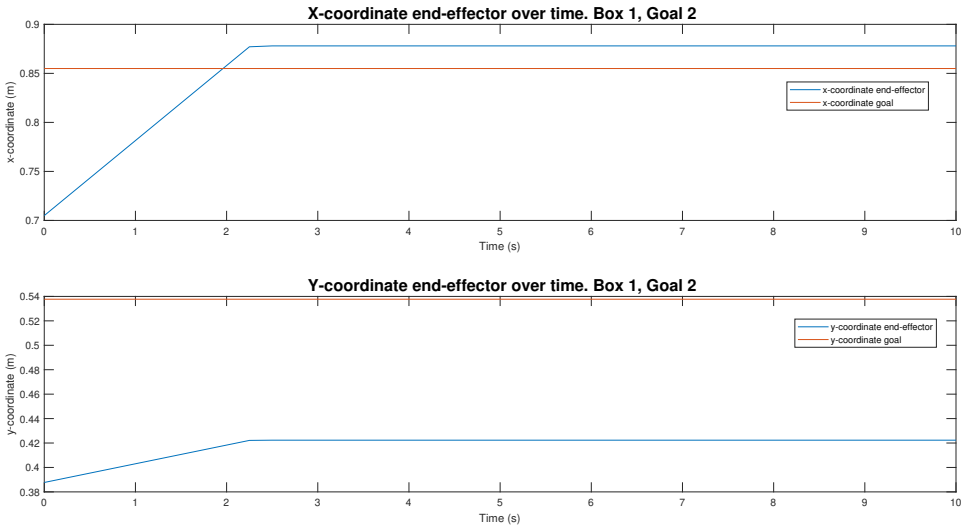


Figure 3-5: Typical push observed for box 1 with goal 2. The x-coordinate of the goal is overshot and the y-coordinate is not attained.

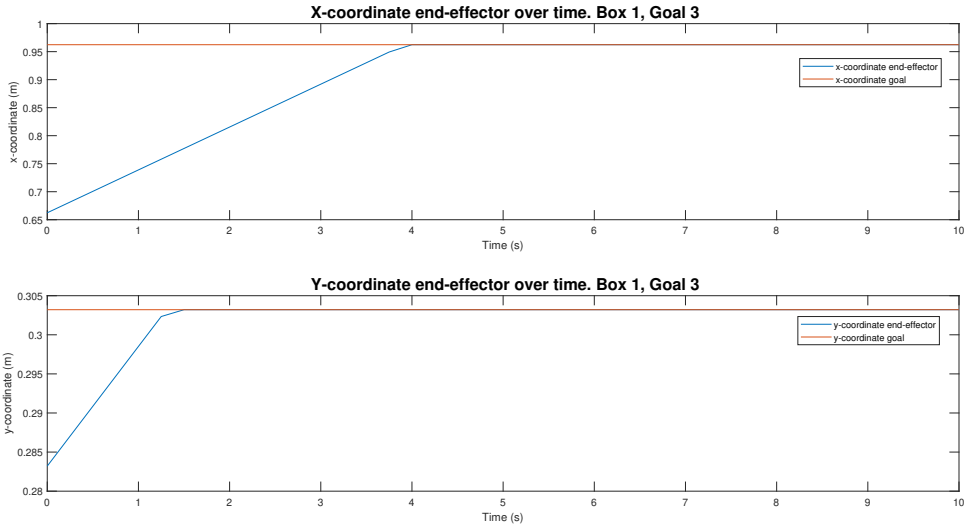


Figure 3-6: Typical push observed for box 1 with goal 3. The x- and y-coordinate are reached successfully.

and the motion stays within the motion cone.

The stances that were used per goal were generated using the method as described in Chapter 2. For the results of Tables 3-1, 3-2, 3-3, and 3-4, we the joint angles for the base stances from which we generate new stances are shown in Table 3-5. For illustration, Figure 3-8 depicts the base stance used for goal 3. All three base stances had similar configurations, where the two hind legs are located behind the body and are spread broadly.

We used normal distributions for each angle with a standard deviation of 8.6 degrees and

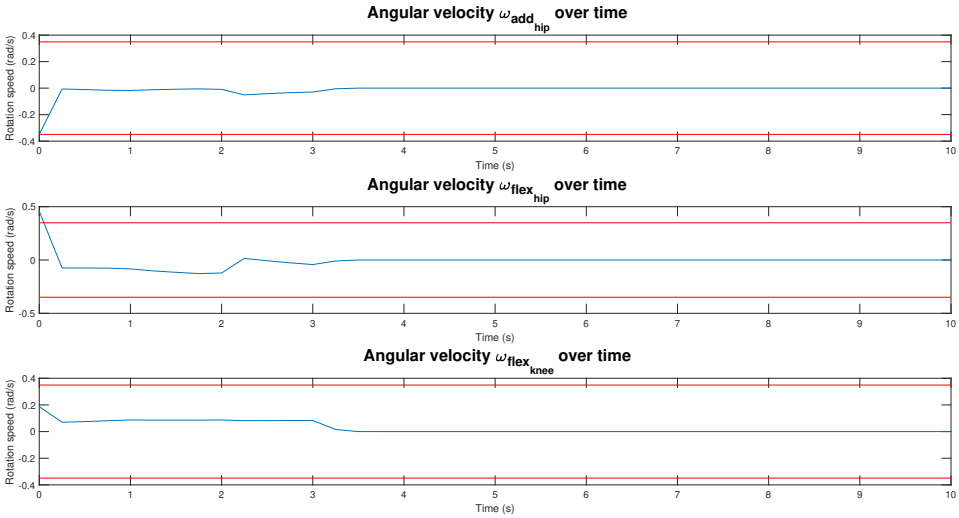


Figure 3-7: Rotation speeds of the joint of the pushing limb. As seen, the maximum rotation speed, visualized by the red lines, for the flexion of the hip is violated at the first time step.

generated 100 stances per push. When the standard deviation was increased to above 15 degrees, performance declined noticeably, mostly in the rates at which the center of mass or the moment of pushing caused an imbalance. It is advised to stay fairly close to the hand-selected base stance in order to preserve robustness.

	Goal 1	Goal 2	Goal 3
θ_{ah_1}	$-33.2°$	$-15.8°$	$15.2°$
θ_{fh_1}	$-39.3°$	$-30.8°$	$-25.5°$
θ_{fk_1}	$142.7°$	$134.6°$	$135.8°$
θ_{ah_2}	$-2.61°$	$19.5°$	$-7.28°$
θ_{fh_2}	$-48.0°$	$-73.2°$	$-32.3°$
θ_{fk_2}	$104.3°$	$125.3°$	$73.9°$
θ_{ah_3}	$-0.264°$	$23.6°$	$38.9°$
θ_{fh_3}	$-96.3°$	$3.9°$	$-19.3°$
θ_{fk_3}	50.0	$25.2°$	$82.5°$
θ_{ah_4}	$15.9°$	22.4	$-10.0°$
θ_{fh_4}	$-110.0°$	$-46.1°$	$-29.6°$
θ_{fk_4}	$76.8°$	$37.0°$	$73.3°$

Table 3-5: Joint angles per goal from which we generate new stances.

3-3-4 Improving Performance

Our method has a clear limitation: because we only work with sticking contacts, the physics of pushing limits the range of goals we can reach fairly severely, in some cases more so than others. One way that will allow us to improve performance, is choose where to place the pusher on the object based on the goal location, rather than always placing it at the center of the face we are pushing against. The motion cone is in part shaped by the placement of

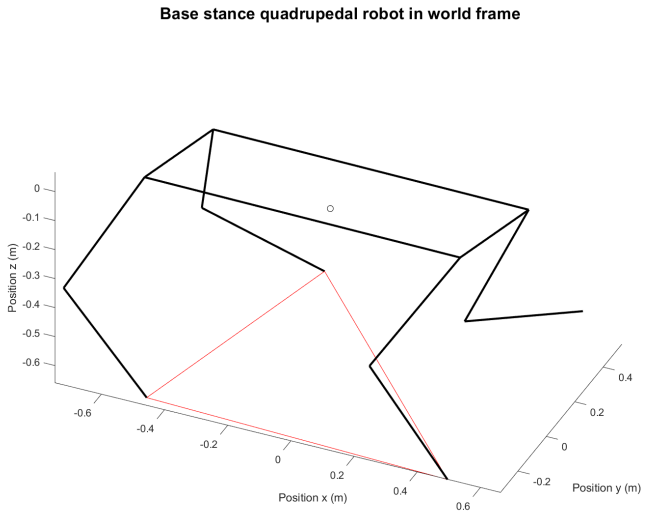


Figure 3-8: Base stance used for goal 3.

the pusher on the object and if we adopt the incorporation of different points from which we push, we can keep all other facets of our optimization problem and approach identical as it is entirely compatible with our previous work.

In Figure 3-9 we can see what influence displacing the center of the pusher with 7.5 cm to the right of the center of box 1 from the simulations has. Though the motion cone becomes more narrow, meaning that the range of velocities we can employ whilst keeping the contact sticking is smaller, it does make for a larger angle to the left of the object, meaning that goals further left than the motion cone that results from a contact of the pusher at the center of the object can reach, can be reached more adequately by displacing the pusher to the right of the box. This principle can thus effectively increase the range of our controller.

We implement this principle by constructing 101 different motion cones with the center of the pusher evenly displaced from the left edge of the object to the right edge. If a number of the generated motion cones contain within them the angle the goal location makes with the x-axis, then displacements belonging to those motion cones are eligible as the final choice. From those displacements, we choose the displacement from the center of the object that has the goal angle closest to the center of its motion cone. If none of the generated motion cones capture the angle needed to push accurately towards the goal, we choose the motion cone whose angle differs the least from the goal's angle with the x-axis. It is important to note that this approach would likely need to be toned down in the real world, as the motion cone may become very narrow, making the pushing motion more prone to slipping contact.

To evaluate the impact that this amendment to the controller has, we look at the performance of the controller steering boxes 1 and 4 towards goals 1 and 2, and box 2 towards goal 2. We made this choice, because said boxes cannot reach these goals accurately with our previous approach, as the angle the goal makes with the x-axis is larger than that of the motion cone. We will also examine the performance of a last box, box 5 on goals 1 and 2. Its properties are:

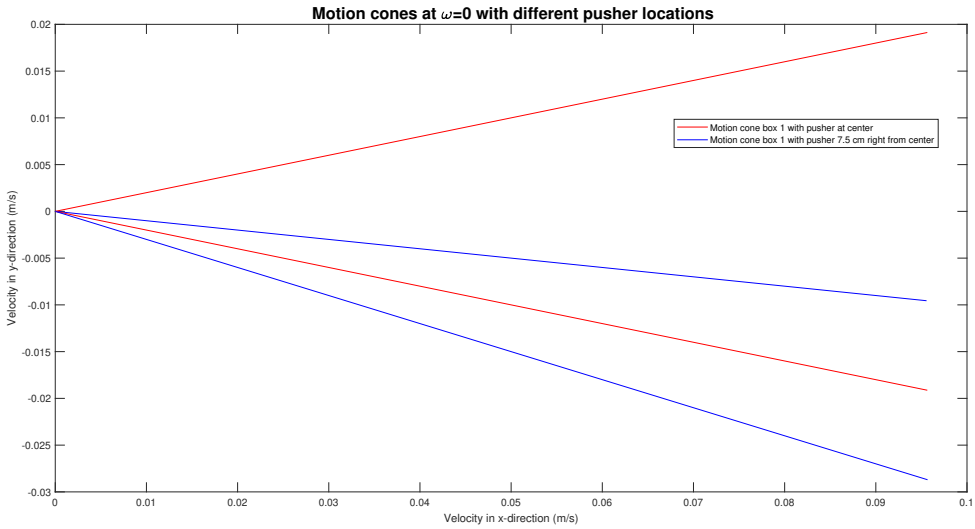


Figure 3-9: Section of the motion cones where rotation speed is 0 rad/s for different pusher contact points. As can be seen, moving the pusher opens up new opportunities.

- Box 5: 50 cm by 50 cm by 50 cm, $\mu_{pusher} = 0.8$, $\mu_{contact} = 0.5$, $m_{box} = 0.5$ kg. Motion cone angle with pusher at center: $\gamma \approx 11.3°$.

Box 5 has the same motion cone angle as box 1 when the pusher is placed at the center of the face we push against, despite the difference in μ_{pusher}. Although the motion cones of boxes 1 and 5 with the pusher at the center do look different overall, the sections of these motion cones at a rotation speed of 0 rad/s are identical. This means that the performance of the controller pushing the boxes towards goals 1, 2, and 3 with the pusher at the center of the pushing face of box 5 would be identical to that of box 1 as portrayed in Table 3-1, where goals 1 and 2 cannot be attained accurately. We will see that the improvement to the controller will make a significant difference between the two boxes, exploiting the fact that box 5 has a higher value for μ_{pusher} than box 1.

Tables 3-6, 3-7, and 3-8 show the results of our improved control method. The results are mixed, as the improved method is does not produce a better result for all objects and goals. In Table 3-6 we can directly compare boxes 1 and 5, which perform identically when we do not allow the pusher to be located on different locations the pushed face of the object. Because of new method, both boxes can be pushed towards goal 1 within millimeters, decreasing the total distance to the goal by 81% for box 1 and 91% for box 5. Goal 2 tells a different story, however, as the controller is much better at pushing box 5 towards goal 2 than box 1. Where the controller gets box 1 on average 10.1 cm removed from the goal, an improvement of only 14%, box 5 is removed only 2.42 cm from the goal, an improvement of 79%. These results underline the importance of designing a pusher with a high coefficient of friction with most materials, as it greatly improves the controllers ability to push away objects given the improved control method.

Tables 3-7 and 3-8 also demonstrate the importance of a high value for μ_{pusher}, as we do not seem great improvements for either box 2 or box 4. The controller's performance on box 2

seemingly does not improve at all with the new approach, implying that objects with wider pushing faces benefit more from this approach than smaller ones as smaller pushing faces hamper how much the pusher can be displaced from the center of that face. Similarly, box 4 only gets 8% closer to goal 1 and sees no improvement on goal 2.

The improvement to the control method by allowing the quadrupedal robot to push from location on the pushing face of the object other than the center, has mixed results. It proves to be very effective at increasing the range of goals that can accurately be attained when the coefficient of friction between the object and the pusher is sufficiently high, but is hampered in its impact at lower coefficients of friction. This approach also produces more meaningful improvements with larger objects.

Furthermore, when the controller is employed in more complex simulation environments or in real world applications, we might want to limit the placement of the pusher to some degree, as the motion cone becomes more narrow towards when moving away from the center of the face we push against. This would mean that slipping between the pusher and the object becomes a greater risk, which is something we want to avoid.

Box 1	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)
Goal 1	$2.88 \cdot 10^{-3}$	0.00224	0.00229
Goal 2	0.0289	0.0964	0.101
Box 5			
Goal 1	$7.48 \cdot 10^{-7}$	0.00134	0.00134
Goal 2	0.0151	0.0189	0.0242

Table 3-6: Results simulation for boxes 1 and 5 with altered method.

Box 2	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)
Goal 2	0.0289	0.0964	0.101

Table 3-7: Results simulation for box 2 with altered method.

Box 4	$\overline{d_x}$ (m)	$\overline{d_y}$ (m)	$\overline{d_{goal}}$ (m)
Goal 1	0.00303	0.0303	0.0305
Goal 2	0.0134	0.134	0.134

Table 3-8: Results simulation for box 4 with altered method.

3-4 Conclusion

In this Chapter we discussed the topic of pushing after stance selection. First we addressed the mechanics of pushing. We opt for abstracting away the dynamics of pushing by approaching the pushing motions as quasi-static and constraining the movement of the end-effector to remain within a section of the motion cone, which ensures that the contact of the end-effector and pushed object sticks.

We performed an experiment with a robot arm to see if the motion cone that we construct matches reality and we saw that for velocities up to at least 0.08 meters per second, the motion cone predicts the movement as accurately as it does in literature, confirming that it can serve as the theoretical basis of a successful controller.

We then discuss how we formulate the MPC problem and how it is solved. It performs as one would expect in simulation. If the goal is contained within the motion cone, the controller gets the end-effector close to the goal and if the goal is outside of the motion cone, the end-effector moves along the edge of the motion cone that is closest to the goal.

This method is then improved upon by allowing the pusher on the end-effector to be placed anywhere on the face of the object we push against. We observe that this addition is particularly effective when the coefficient of friction between the pusher and the object is higher, but improvement is limited otherwise.

There exists precedent for non-prehensile manipulation using motion cones and it seems suitable our use case as well. Pushing as a form of object manipulation has fundamental drawbacks, that we cannot escape, meaning that not every goal can be reached with the accuracy we might aspire. There do exist, however, ideas that can be expanded upon in further work that might increase the range of goals that can be reached through pushing. This will be discussed in Chapter 4.

All in all, through the use of motion cones we construct a planner that abstracts away the complicated dynamics of pushing and finds a trajectories along or within the bounds of these motion cones to push an object as close to the goal as the physics of the situation allow, with sticking contact.

Chapter 4 - Discussion

In Chapter 1 we defined the problem this thesis seeks to answer as:

- ▷ **Design a controller for a quadrupedal robot that can push away various objects obstructing its path.**

Chapters 2 and 3 have left us with a coherent strategy for pushing various objects with a quadrupedal robot, but it is wise to provide the work we have performed with rigorous critique and see what the impact is of the choices we made and what we may have chosen to do differently.

In Sections 4-1 and 4-2, we look at the choices we made in Chapters 2 and 3 and discuss if these were fair and reasonable and what other routes we could have taken. Section 4-3 discusses the future work that could be performed to come closer to a controller that can be employed in our use case. Finally, in Section 4-4 we give a brief summary and our conclusion to the thesis.

4-1 Discussion Chapter 2

Chapter 2 discusses the topic of stance selection and does so by first giving a number of assumption that we then build from in Section 2-1. We should ask ourselves whether these assumptions are reasonable.

The set of assumptions pertaining the dimensions of the quadrupedal robot and its range of motion are difficult to argue against and easily changed without having to alter the stance filtering algorithm 1, though the assumption that all the weight is evenly distributed for all body parts will not hold up for real quadrupedal robots. The assumption that the robot works only on flat, level ground is fair in our use case, though many quadrupedal robots are designed to be able to walk on a slope as well. Some level of scrutiny is necessary when evaluating the assumption that we know the properties of the object we wish to push, as this is not a given in our use case.

First we will address the assumption that all components of the robot have equally distributed mass. It should first be noted that, while this assumption will not be true for any commercially available quadrupedal robot, successful stance selection does not hinge on this assumption being truthful. This assumption exists so we can calculate the location of the center of mass, and we can still do this even if the distribution of mass for each part of the robot is uneven, we would just need to estimate this distribution. In literature [25] the location of the center of mass of a quadrupedal robot, Boston Dynamics' Spot equipped with a gripper arm, is estimated by approximating the total mass through a number of point masses at different locations on the robot. If balance becomes an issue in future applications, such an approach should be suitable in our use case.

We assume that we know the properties of the objects the robot pushes. This assumption is ubiquitous when working with a quasi-static model and has produced successful pushing strategies [23, 26, 27, 28], but is not likely to always hold in our use case. In Section 4-3, we will address the fact that we will not always know the properties of the object.

Section 2-2-1 is relatively straightforward, but our choice for ANFIS networks to solve the inverse kinematics problem in Section 2-2-2 should be reevaluated. As we explain in Section

2-2-2 it performs well and swiftly enough to be used real time applications. Furthermore, as we see in Chapter 3, the stances that are ultimately chosen generally perform well with regards to staying balanced. Ultimately, we feel that the method of solving the inverse kinematics problem suits the use case and will remain appropriate in later development.

The filters present in algorithm 1 are not entirely comprehensive, as mentioned in Chapter 2. We do argue, however, that in our application, where we generate a final stance from an appropriate base stance ensures that all the oddities that are not accounted for by the filters, do not occur. We have found this to be true in practice as well, since the stances we select never deviate far from the base stance when the distribution is properly chosen. If we allow the stance generation to stray further from the base stance, issues do arise, so it is important we choose an appropriate base stance and to generate joint angles that do not deviate far from said base stance.

Chapter 2 provides us with a fairly robust and comprehensive method of generating stances that appear to perform well during the pushing task, as we can see in Chapter 3. There are some open-ended questions regarding stance selection that the chapter does not address, because they are outside of the scope of this thesis. We will look into the most pressing questions on stance selection in Section 4-3.

4-2 Discussion Chapter 3

We come to a model predictive controller in Chapter 3, that behaves in a way we would expect: if the goal location lies within the motion cone, the end-effector is navigated to the goal accurately, and if the goal location lies outside of the motion cone, then the end-effector moves along the edge of the motion cone that bring it closest to the goal. Furthermore, it can exploit the fact that motion cones are different depending on the location of the pusher with respect to the object to increase performance. There are some important remarks to be made, however, regarding the choices we made when coming to a controller.

The usage of motion cones allows for computationally efficient planning and it is easily integrated within the MPC structure as a constraint, but in our application it does mean that the range of goals where an object can be pushed accurately is often limited to a small range. One could argue that our use case does not demand a controller that can push an object to a wide range of locations accurately, since it only needs to move objects out of its way so the robot may move these objects. It still stands, however, that our planner is limited, particularly in cases where the coefficient of friction between the pusher and object is low, and we should investigate ways with which we can expand its capabilities.

Another fundamental limitation of the controller is that it needs the construction of a motion cone in order to function. This is not a problem when we are faced with pushing an object that has flat faces that the end-effector can be placed against, but we cannot reasonably expect that to always be the case. There are a great many household objects that might obstruct a quadruped that realistically can be assigned a motion cone, such as nightstands, storage boxes and closets, but objects such as chairs and tables might prove incompatible with our method of control. We cannot guarantee that we will always be able to circumvent objects with geometries that are not as easily typified by a motion cone.

Objections may also arise concerning the assumptions we make regarding the various ways we control joints. We assume that the joints that are not involved in the pushing motion

stay still during the pushing motion and that the joints of the pushing limb can be controlled smoothly and are not limited in the torque they can provide. In the context of delivering a proof of concept this can be overlooked, but in real-life applications we must consider the control of all joints during the pushing motion in a more realistic manner.

While Chapter 3 describes a controller that can move the end-effector of a quadrupedal robot to a goal in such a way that the object it is pushing reaches that goal as well, we see that the controller is limited in the goals it can reach accurately, can handle only a limited amount of geometries, and assumes that the joints behave exactly as we want them to do. If we want to progress the controller to real life settings, we must look into these problems. We will do so in Section 4-3.

4-3 Future Work

In Sections 4-1 and 4-2 we identified a number of issues that should be resolved if our controller is to function in real life. Furthermore, there exist some topics that fall outside of the scope of this thesis that one might suspect to be relevant to our use case. We will look into these topics in this Section and provide suggestions on how to strengthen the controller we designed, if available by providing examples from literature.

4-3-1 Object Uncertainty

As we identified, we assume that we know the properties of the objects we wish to push and we only address objects with cuboid geometries. We will have to account for uncertainties and more complex object geometries.

One method that may prove useful when dealing with model uncertainty, is to adopt a control method that specifically accounts for model uncertainty. One such example that is found in literature is named Ensemble Model Predictive Path Integral Control, or EMPPI[29, 30]. This is an MPC method that specifically accounts for uncertainty by generating a distribution for all uncertain parameters within a certain range, which decreases as an action is performed and the controller understands the system better. EMPPI is compatible with real time uses and has successfully completed tasks involving object manipulation, among which a task where a block with unknown mass and an unknown coefficient of friction with the ground surface is pushed to a target by a robot arm.

Constructing motion cones for objects with irregular geometries is a subject that is unexplored in literature and would require additional research. Most household objects that cannot reasonably be described as cuboid, do contain flat faced surfaces on some points of their geometry, such as tables on their legs and top, though they are typically thin and located in unfortunate positions when pushing is considered. We recommend investigating if motion cones can typify the behavior of such objects as well as it does for objects with a cuboid geometry. If that is indeed the case, then our proposed controller can be expanded to most household objects, if not, then we must either concede that not all objects can be pushed as we desire or search for different control methods.

4-3-2 Increase Push Range

One of the major flaws of our approach has to do with the range of goals that the controller can accurately attain. If we wish to expand this range while holding on to our method of only allowing stable pushes to push away objects, there is one improvement we can make that

would reliably increase the range of goals we could reach: rotating the object into a more advantageous orientation.

The reason why rotating the object is a powerful way of increasing the range of goals we can accurately reach, is because rotation can allow us have the object face the goal in such a way that the object's motion cone shifts to face the goal location. There are two ways to achieve this and both are warrant additional research.

The first one is to allow the pusher at the end-effector to induce rotation of the object by supplying it with a wrist joint. A wrist joint gives the robot the option to give objects a more desirable position for pushing towards their goal. Providing the end-effector with a wrist joint is no small feat, as it would have to be integrated into the hardware of the robot and not be detrimental to the locomotion of the robot, which is a far more important function than the ability to push objects.

We could also opt for multiple pushes, where the first push, or multiple pushes, intentionally induce slip between the pusher and the object, which is what we explicitly try to avoid in our current approach. The slipping motion would cause the object to rotate. If we can do this in a controlled manner, we could rotate the object in a favourable way and perform a push where the contact sticks afterwards and more accurately reach our goal.

We should also consider to look into achieving goal locations that are outside of the range of motion for the robot. This would necessitate that the robot sequentially pushes and moves, until the goal is reached. This could be combined with the ability to increase the range of goals we can achieve in a single push, as described in the text above, to greatly increase the amount of space the robot can push objects towards.

4-3-3 Joint Control

The limbs of quadrupedal robot can be categorized into three different types: prismatic limbs, articulated limbs, and redundant articulated limbs [31]. In quadrupedal robots of the size that we are considering in this thesis, articulated limbs are the most common type. They typically have three degrees of freedom and are modeled to have a mammal-like configurations. The joints in these limbs differ from robot to robot, and are not known for every quadrupedal robot. Some examples of joints used in quadrupedal robots include the series elastic actuators [32] found in the ANYmal quadruped robot [33], the proprioceptive actuators in the Cheetah3 [34], and the motor driven joints in the Jueying robot [31].

In order to be able to smoothly track the joint angles the controller wants the robot to achieve, an accurate model of the joints is needed. As most joints are intricate in their designs and reliant on a variety of parameters, this is not a trivial task. We can opt to model these joints analytically [32] or to approximate the joints through a data-driven model [35]. In either case, we would need to know exactly what sort of joints we are working with and investigate what level of accuracy is needed in order to come to adequate results.

4-3-4 Disturbance Rejection

Disturbance rejection is a topic that bears importance to the performance of the controller and one that typically helps define the quality of the controller. We would need to expand our controller for disturbances that cause the contact between the object and the end-effector to break, as this would mean that the robot needs to either reposition itself entirely or extend its arm in such a way that contact is re-established. Smaller disturbances that do not cause

contact to break can be incorporated into the controller as long as the motion cone changes accordingly.

In our current simulations we only track the position of the end-effector and, because all velocities are contained within the motion cone, we assume that the velocities of the object are identical. This approach makes a thorough analysis of disturbance rejection impossible, because we would need to track and disturb the motion of the object. We mention the importance of further simulation and experimentation in Section 4-3-6.

4-3-5 Achieving and Retaining Stance

We assume that the robot can move towards the object and take on the stance that is produced by algorithm 1 and considered optimal out of all remaining stances. This is not a trivial task. It can be assumed that any commercially available quadrupedal robot can move towards the object of interest, but getting into a particular stance is a more difficult task, even if we have full control over the joint angles. Readjusting the joint angles in such a way that the robot remains balanced throughout the entire process is something that warrants further research. Similarly, keeping the robot balanced and the joint angles constant throughout the pushing motion is a task that seems challenging.

As such behavior is likely difficult to achieve through entirely handcrafted methods, one interesting method we could consider is a deep reinforcement learning approach in a similar vein as is presented in [35]. In their paper, the researchers present a deep reinforcement learning approach where a quadrupedal robot learns to move into an upright position after falling using three neural networks. The behavior is learned entirely within a physics simulator, but has a success rate of over 97% in real world experiments. In our use case, one network could be trained to achieve the appropriate stance prior to the push and another to maintain the stance during the push.

4-3-6 Experimentation

Ultimately, the most important work to be done in the foreseeable future is to test the controller in a more realistic simulation or in the real world. We can add to the controller by looking into the other topics discussed in this Section, but the thesis does deliver us with a controller that has the ability to push objects to their intended goals and it is imperative that we confirm the validity of this approach employing more complete simulation environments and the real world.

Gazebo [36] and MuJoCo [37] are physics engines that are suitable for our use case and could provide us with more insight into our control method if used. As seen in Chapter 3, the motion cone predicts stick and slip with an accuracy slightly above 80%, which was consistent with literature, and sufficient for other controllers to succeed in object manipulation tasks. We can justify our approach through these findings, but physics engines would be a risk-free method of further strengthening our justification.

Our proof of concept would turn into a real practical application if we manage to validate the work done in this thesis in real life. The KUKA arm of Chapter 3 would make a good candidate for more experiments in the real world, though it would need to be adapted to be able to be controlled by the MPC controller we designed, where it can smoothly follow the joint angles that the controller prescribes. There are more obstacles to moving away from simulation, as we would need to track the object to accurately estimate the motion cone per

time step and the optimization problem given by equation 3-14 would have to be adapted, as the x-, y- and z-coordinate are determined differently and it has seven joint angles that can be controlled as opposed to three.

4-4 Conclusion

We tasked ourselves with designing a controller for a quadrupedal robot that pushes away objects. Having a quadrupedal robot use its limbs to manipulate objects is an unexplored topic in current literature, meaning we needed to devise a strategy ourselves. This strategy largely consists of two parts: finding an appropriate stance and planning and performing the push.

The stance selection stage consists of stance generation, stance filtering and a final stance selection. We generate stances by first taking a base stance with twelve user-defined joint angles and taking a normal distribution over each joint angle, creating new stances. We take this approach to prevent the need to generate an excessive amount of stances, which would mostly be useless.

Each stance is then filtered by passing it through a number of tests, all of which need to be passed for the stance to be considered as the final stance. These tests rely on us first solving the stances' forward and inverse kinematics. The forward kinematics are solved mostly through basic geometry and some rotation matrices, which help us determine the position of all of the robot's limbs and its orientation with respect to the ground. The inverse kinematics are solved using three ANFIS networks, which estimate the three joint angles of the pushing limb at the goal location. Solving the forward kinematics helps us create tests that assess if the robot's starting position is sensible and balanced, whereas solving the inverse kinematics determines if reaching the goal location can be reached, and, if so, whether it can be reached without jeopardizing the robot's balance.

All stances that make it through the filtering process are eligible to be the final stance. The selection is made by comparing the total displacement of the joint angles to that of the robot's resting stance. The stance that has the smallest total joint angle displacement is considered to be the optimal candidate and selected as the final stance.

The final stance provides us with a starting point from which we can push. We use the principle of motion cones to come to a sensible controller. Modeling pushing dynamics is complicated and computationally expensive, so we opt for a quasi-static approach that allows us to neglect the dynamics of pushing. We model the mechanics of pushing through two different concepts: the limit surface and the generalized friction cone. The limit surface models the interaction between the object we wish to push and the surface it rests upon, and the generalized friction cone models the interaction between the pusher and the object. Together, these two concepts allow for the construction of the motion cone: a set of object twists that can be achieved with sticking contact, meaning contact between the pusher and object where the two do not move relative to each other.

Through an experiment employing a robot arm, we test the validity of our approach and see that our results are similar to findings in literature; the motion cones predict sticking and slipping motion based on the pusher's movement with an accuracy slightly higher than 80%. This result is encouraging, because motion cones with similar accuracies have formed the basis of other successful controllers.

The model predictive controller we design solves a nonlinear optimization problem over each time step, where the cost function is the sum of the distances between the goal location and the current location of the end-effector in the x- and y-direction over the finite time horizon and the decision variables are the joint angles of the pushing limb. We constrain the optimization problem in several ways: the joint angles cannot exceed their physical maximum and minimum, they cannot turn faster than the hardware allows them to, the z-coordinate of the end-effector has to stay between the ground level and the height of the object, and the velocities in the x- and y-direction need to comply with the motion cone.

We have tested the ability of the MPC controller to push a variety of cuboid objects, or boxes for simplicity, to three different goals in simulation. Our controller performed in the way one would expect given our approach. If the angle of the push needed to achieve the goal is contained within the motion cone and the goal is within the reach of motion of the pushing limb, then the controller reaches the goal accurately. By shifting the position of the end-effector on the object, we can influence the shape of the motion cone and, in cases where the coefficient of friction between pusher and object are high, expand the range of goals we can achieve significantly. If the angle of the push is too large and falls outside of the range that the motion cone allows, then the controlled push is on the edge of the motion cone.

The limitation that we cannot push outside of the motion cone accurately is fundamentally related to the physics of the problem we try to solve and our method of solving that problem. There are several ways to build on the work done in this thesis and improve upon the current controller. We can increase the range of goals that we can reach accurately by shifting the object's orientation. To achieve this, we would have to introduce a wrist joint that can induce rotations of the object or adopt a strategy where we purposefully slip along the object to turn it in a more favorable orientation.

There are other improvements that we consider. In our approach, we assume that we know the properties of the object we push, when we typically will not have such knowledge. We could choose to adopt a method of MPC that accounts for unknown variables. Furthermore, we should look into methods that help us reliably achieve and maintain the stance we select for with the stance selection algorithm and, dependant on which robot we employ, how we might model the joints most accurately. Finally, the controller in its current form should be tested rigorously in environment that more closely emulate the use case.

The contribution of this thesis is that it provides a solution to a problem that has not been explored in scientific literature. The filtering algorithm we use to determine the most appropriate stance among all candidates, algorithm 1, is, to our knowledge, unique. Though it does not rely on any theoretical concepts specifically developed for our use case, the method with which we combine different concepts to solve the stance selection process is novel.

The same holds true for our implementation of MPC. We do not develop any unique methods, but the optimization problem we formulate is one that handles a problem that has no precedence in literature. Furthermore, it is an approach that can be expanded in various ways to make for a more capable controller without needing to alter any of the work performed in this thesis.

Bibliography

[1] B. Dynamics, "About spot." https://dev.bostondynamics.com/docs/concepts/about_spot, 2023.

[2] S. Zhang, X. Rong, Y. Li, and B. Li, "A composite cog trajectory planning method for the quadruped robot walking on rough terrain," *International Journal of Control and Automation*, vol. 8, no. 9, pp. 101–118, 2015.

[3] A.-V. Duka, "Anfis based solution to the inverse kinematics of a 3dof planar manipulator," *Procedia Technology*, vol. 19, pp. 526–533, 2015.

[4] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *IEEE transactions on systems, man, and cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[5] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*, vol. 200. Springer, 2008.

[6] C. P. Janssen, S. F. Donker, D. P. Brumby, and A. L. Kun, "History and future of human-automation interaction," *International Journal of Human-Computer Studies*, vol. 131, pp. 99–107, 2019.

[7] S. Šabanović, "Robots in society, society in robots: Mutual shaping of society and technology as a framework for social robot design," *International Journal of Social Robotics*, vol. 2, no. 4, pp. 439–450, 2010.

[8] J. Stüber, C. Zito, and R. Stolkin, "Let's push things forward: A survey on robot pushing," *Frontiers in Robotics and AI*, p. 8, 2020.

[9] A. Li, Z. Wang, J. Wu, and Q. Zhu, "Efficient learning of control policies for robust quadruped bounding using pretrained neural networks," *arXiv preprint arXiv:2011.00446*, 2020.

[10] B. Karlik and S. Aydin, "An improved approach to the solution of inverse kinematics problems for robot manipulators," *Engineering applications of artificial intelligence*, vol. 13, no. 2, pp. 159–164, 2000.

[11] R. Pérez-Rodríguez, A. Marcano-Cedeño, Ú. Costa, J. Solana, C. Cáceres, E. Opisso, J. M. Tormos, J. Medina, and E. J. Gómez, "Inverse kinematics of a 6 dof human upper limb using anfis and ann for anticipatory actuation in adl-based physical neurorehabilitation," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9612–9622, 2012.

[12] C.-H. Chen and D. S. Naidu, "Hybrid control strategies for a five-finger robotic hand," *Biomedical Signal Processing and Control*, vol. 8, no. 4, pp. 382–390, 2013.

[13] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[14] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "Planar in-hand manipulation via motion cones," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 163–182, 2020.

[15] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "In-hand manipulation via motion cones," *arXiv preprint arXiv:1810.00219*, 2018.

[16] S. Goyal, *Planar sliding of a rigid body with dry friction: limit surfaces and dynamics of motion*. PhD thesis, Cornell University Ithaca, NY, 1989.

[17] N. Xydas and I. Kao, "Modeling of contact mechanics and friction limit surfaces for soft fingers in robotics, with experimental results," *The International Journal of Robotics Research*, vol. 18, no. 9, pp. 941–950, 1999.

[18] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 557–572, 1996.

[19] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback.," in *IROS*, vol. 1, pp. 416–421, 1992.

[20] J. Zhou, Y. Hou, and M. T. Mason, "Pushing revisited: Differential flatness, trajectory planning, and stabilization," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1477–1489, 2019.

[21] J. Shi, J. Z. Woodruff, P. B. Umbanhowar, and K. M. Lynch, "Dynamic in-hand sliding manipulation," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 778–795, 2017.

[22] M. Erdmann, "On a representation of friction in configuration space," *The International Journal of Robotics Research*, vol. 13, no. 3, pp. 240–271, 1994.

[23] K. M. Lynch, "The mechanics of fine manipulation by pushing.," in *ICRA*, pp. 2269–2276, Citeseer, 1992.

[24] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pp. 800–815, Springer, 2020.

[25] S. Zimmermann, R. Poranne, and S. Coros, "Go fetch!-dynamic grasps using boston dynamics spot with external robotic arm," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4488–4494, IEEE, 2021.

[26] D. J. Cappelleri, J. Fink, B. Mukundakrishnan, V. Kumar, and J. C. Trinkle, "Designing open-loop plans for planar micro-manipulation," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 637–642, IEEE, 2006.

[27] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[28] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "Path planning for pushing a disk using compliance," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 714–720, IEEE, 2005.

[29] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2864–2871, 2020.

[30] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control." https://sites.google.com/view/emppi, 2020.

[31] Y. Zhong, R. Wang, H. Feng, and Y. Chen, "Analysis and research of quadruped robot's legs: A comprehensive review," *International Journal of Advanced Robotic Systems*, vol. 16, no. 3, p. 1729881419844148, 2019.

[32] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 1, pp. 399–406, IEEE, 1995.

[33] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 38–44, IEEE, 2016.

[34] P. M. Wensing, A. Wang, S. Seok, D. Otten, J. Lang, and S. Kim, "Proprioceptive actuator design in the mit cheetah: Impact mitigation and high-bandwidth physical interaction for dynamic legged robots," *Ieee transactions on robotics*, vol. 33, no. 3, pp. 509–522, 2017.

[35] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *arXiv preprint arXiv:1901.07517*, 2019.

[36] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[37] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.