Footstep Planning for Legged Robots using Mixed-Integer Programming

James Thomson

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2023

Abstract

For a legged robot to achieve its goals, it must be able to navigate through its walkable space from an arbitrary start position to a desired end position. To make this navigation possible, it must plan a sequence of footsteps which start and end at those desired points. However, if one wishes this plan to be optimal, without backtracking, it must be able to "look ahead". For example, a plan must consider possible future steps and foot placements to ensure that the robot avoids situations where it is forced to retrace its path or make inefficient movements. This work will focus on the generation of a sequences of footstep positions which are globally optimal over an environment through Mixed-Integer Programming (MIP). Given the combinatoric complexity of discrete optimisation, we propose an offline approach which abstracts the environment into a graph which encodes paths through the environment. This helps us re-plan online in the event of unexpected footstep placements, which can be caused by a number of factors and uncertainties during locomotion, without performing any additional combinatoric optimisation. We investigate the effectiveness of this approach compared to the state-of-the-art MIP approach in certain re-planning contexts, and present data which shows that it greatly reduces the time taken to create a new plan given an unexpected footstep position part-way through locomotion.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*James Thomson*)

Acknowledgements

I would like to thank Steve Tonneau for his invaluable guidance with the project.

Table of Contents

Chapter 1

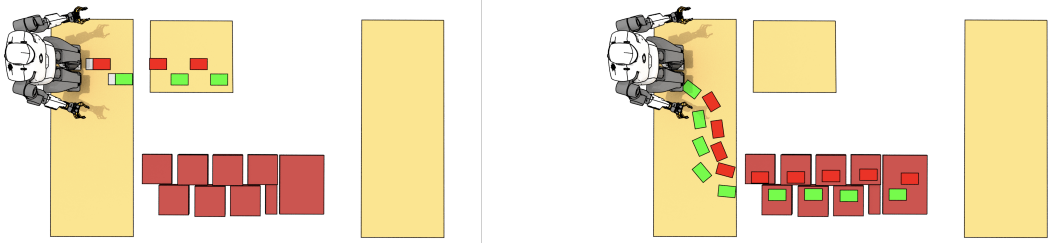
Introduction

1.1 Background and Motivation

Contact planning is a critical aspect of robotic manipulation and locomotion, enabling robots to interact with their environment effectively and safely. In recent years, the demand for robots with advanced contact planning capabilities has grown significantly, driven by applications in manufacturing, logistics, healthcare, and service industries [8] [5]. Furthermore, sources of motivation to advance the field of legged locomotion are endless, just as are the uses for a human's own two feet. The main promise of legged locomotion is to provide a substitute to humans and animals in performing menial or dangerous tasks without the additional accommodations required by wheeled robots [18]. For instance, locomotion over uncertain terrain – such as in disaster rescue – is an especially difficult problem for robots [36]. If we can reduce the robot's time spent planning, the energy and time saved will potentially be the difference between life and death for both humans and animals in disaster situations. By improving a robot's ability to create a plan to navigate a given environment, it can reach victims faster and offload the risk from human first responders, increasing the chances of survival for those in need. On a less deadly and more general note, an increase in efficiency of automata in a production environment such as a factory provides an opportunity to reduce operational costs and increase productivity [20].

For an autonomous robot to navigate its environment, it must be able to generate a plan to move from point A to point B. In the case of a legged robot, this plan will most likely involve the locations at which its limbs make contact with the ground. These locations, which we call *contact points*, will be the main focus of this dissertation. Given a sufficiently complex environment, a robot planning its route one step at a time is bound to run into a point at which it must backtrack before reaching its target. To prevent this, our robot needs an amount of "foresight" which allows it to detect these dead ends and local minima to prevent the generation of naïve plans such as in 1.1a. This problem of global path planning remains an open area of research [32], with various approaches and ways to abstract the environment to create a tractable problem, such as Probabilistic Roadmap Methods (PRM) [24], or Cellular Decomposition [25].

Going forward, we will use the term *offline* to refer to computations performed before



(a) An example of naïve pathfinding resulting in a contact plan that falls into a local minimum.

(b) An example of a path obtained through global pathfinding methods with “foresight”.

Figure 1.1: [33] An illustration of two paths through an environment of convex regions, the left path reaches a dead end by going straight towards the objective, the right path takes slight detour, resulting in a superior path to the left path.

beginning locomotion, and *online* to refer to those performed after beginning locomotion.

One large roadblock staring down real-world applications of robotics is the uncertainty of the real world [30]. This can manifest as an unexpected impulse on the robot, or a change in the goal or environment part-way through locomotion, among many other things. The former may cause the robot to end up in a different position than we anticipate at some point during locomotion, invalidating the contact plan. In each case, we are required to re-plan a route. With a naïve approach, this may be just as expensive as when we computed a route in the first place and, at a high number of steps, this may be an unacceptable interruption – especially if such interruptions are frequent. Currently, there exist methods in which this is expected and accounted for, such as Model Predictive Control (MPC), in which a rolling optimisation is performed at every step to generate a trajectory that takes into account the current state of the robot and the environment [10]. We do not implement an MPC model. Instead, we are interested in expanding upon some of the ideas laid out by Deits and Tedrake in [31], which we will cover in Section 1.3. If we can create a useful abstraction of the environment offline which can be quickly queried online to generate long, optimal contact plans in real-time, we can mitigate the problem of online re-planning and improve the autonomy of the robot while maintaining optimal solutions. In the latter half of this paper, we will develop this notion by providing a novel offline graph-construction procedure which allows us to capture a near-analytical[1] abstraction of the robot’s ability to move through the environment with respect to the constraints we set and simplifications we make throughout the text.

1.2 Overview of Approach

The essential problem to tackle is the generation of a vector x of contact points in 2-dimensional space, beginning and ending at specific points and subject to a number

[1] It is not completely analytical due to the linear inner-approximation outlined in Section 4.1.1.

of constraints. The majority of these constraints are geometric "bounding" constraints which are linear and form a number of distinct convex regions as shown in 1.1. We call this collection of regions the *environment*. In order to generate a plan which traverses an environment through multiple regions, we must "choose between" each region in the environment for each contact point. Hence, a decision must be made for which region each contact point is in. This introduces a degree of combinatorics to the problem. The critical aspect of this approach is the reduction of this combinatoric complexity. We employ a series of simplifications in order to abstract away a large chunk of what it takes to make a robot locomote in favour of condensing the problem to this single area of focus. To this end, the environment space will consist of 2-dimensional walkable regions in an implied 3-dimensional world. We will work with a mathematical model of a robot which largely ignores dynamics as well as many real-world options for movement such as rotation of the foot, and 3-dimensional planning. We make the quasi-static assumption, which simplifies the robot's dynamics to create a tractable problem. This assumption has been shown to greatly reduce the complexity of a problem while producing useful results [14]. Despite this simplification, the planning problem remains challenging. When planning multiple steps at a time, the complexity of the contact planning problem increases dramatically due to the exponential number of decisions the robot must make.

The goal of this paper is to work on improving the performance of this discretised method through reduction of the combinatoric complexity. The approach taken is an independently developed method of offline environment abstraction in the form of a graph which encodes paths through the environment by mapping paths through the graph to sequences of values on the aforementioned integer variables. In essence, we spend extra time abstracting the environment before locomotion to spend less time computing the path itself, as well as any subsequent paths through that same environment. As a consequence of our blind search to construct the graph, we inevitably encode other paths through the environment. We use these other paths to assist the state-of-the-art formulation in re-planning a path online in less time than it would take otherwise. It is important to note that the coarse approximations we use to construct our model are sufficient, as is shown in the following section.

1.3 Prior work

An "optimal" path in this context is understood to be that which contains the minimal number of steps, as it is commonly used this way in literature [31], [16].

Two common approaches for contact planning are:

- Discretised action set, as seen in [16], [27], and [3]: This approach involves limiting the action space into a finite set of actions. A tree structure is then formed, with branches representing possible sequences of actions. The robot searches through this tree using graph traversal algorithms such as A* [15] or RRT [23] to find an optimal sequence of actions to reach its goal. This approach suffers from the tradeoff between a small and large action set, the former reducing the branching factor of the tree while covering less of the true solution space, and

the latter covering more of the solution space but being more difficult to search [3].

- Continuous planning: This approach considers the continuous nature of the robot's walkable space, allowing for more precise and flexible plans which cover the entire action space. However, this type of model may require non-convex constraints in order to ensure obstacle avoidance, and is not guaranteed to generate globally optimal paths [7].

Hybrid approaches often combine the advantages of both discretised and continuous planning, such as balancing computation time and solution optimality. One such hybrid method we will focus on is the Mixed-Integer Program (MIP) approach proposed by Deits and Tedrake [31]. Specifically, they propose deconstruction of the problem space into a discrete set of continuous, convex regions and use integer variables to assign each footstep to a region. Through this partial discretisation of the action space, we introduce combinatoric optimisation as stated above, in that we must find a combination of integer variables corresponding to the shortest path from the start to the end. On the other hand, we are able to take advantage of the mathematical properties of a convex solution space, which is generally easier to solve for as it does not carry a risk of falling into local minima [4].

It has been shown that the coarse approximation that we will be making for our formulation of the problem is sufficient [31]. Specifically, we can formulate the problem as a single MIP in which reachability is enforced by a convex inner approximation of the robot's reachable space and the environment is decomposed into "convex regions of obstacle-free configuration space and assigning each footstep to one such safe region" [31]. Perception of the environment is not part of our concern in this context, and the development of IRIS – an "algorithm for computing convex regions of obstacle-free space" [9] – makes it evident that this abstraction is reasonable to make, as it is used the same way in the state-of-the-art. Deits and Tedrake propose a Mixed-Integer Quadratically Constrained Quadratic Program (MIQCQP) formulation which generates contact points, accounting for foot rotation and including variables which encourage a minimum number of steps. In this paper, we introduce a Mixed-Integer Quadratically Constrained Program (MIQCP) formulation which does not account for footstep rotation and takes a fixed number of steps as part of its input, among other simplifications. The class of formulation differs in their objectives: quadratic for MIQCQP and linear for MIQCP.

This fixed step number in our formulation would pose a problem with regards to optimal plan generation if not for the offline graph approach we introduce in Chapter 4, as it finds the optimal number of steps and feeds that value to our formulation.

1.4 Thesis aims and achievements

The aims of this dissertation were as follows:

i. Provide an overview of Mixed-Integer Programming, and how it is applied to contact planning in robotics in the state-of-the-art.

ii. Implement a Mixed-Integer Programming formulation in the spirit of the formulation proposed by Robin Deits and Russ Tedrake in their 2014 paper [31], which can generate contact points through an environment.

iii. Contribute and analyse a proposed extension to the state-of-the-art.

Each of these aims were achieved. The contribution made was the graph-based approach introduced in Chapter 4, which was then analysed in Chapters 5 and 6, and was shown to succeed in its goal of assisting the state-of-the-art MIP approach to replan paths faster. All relevant code can be found at `https://github.com/jathoms/optimization-for-footstep-planning/`. Which can hopefully be used for further exploration.

All of the visualisations in this dissertation are screencaps of the live implementation written in Python, generated and displayed using `Matplotlib` [26] and `netowrkx` [28] (among other packages) unless cited as otherwise. Visualisations of the procedures in action are generated in `graph_traversal.py`, and miscellaneous visualisations, including results, are generated by one of the functions in `visualisation_generator.py`.

Chapter 2

Foundations of Methods

When we say "formulation", we are talking about the framing of our contact planning problem as a mathematical model of the problem solvable using optimisation techniques employed by a *solver*. In general, we can categorise these models into one of two camps. Our model can represent an *optimisation problem* or a *feasibility problem*. An optimisation problem involves an *objective function* and *constraints*, the former of which is a mathematical expression that represents the goal or objective we seek to optimise, such as minimising cost or maximising efficiency. The latter, constraints, are a set of conditions or restrictions that the solution must satisfy. These constraints can represent physical limitations, resource availability, or other factors that govern the feasibility of a solution. In the context of our contact planning problem, we do not include an objective function by default, as we value any solution satisfying the constraints equally, and therefore the formulation we provide is a *feasibility problem*, as it only involves constraints. The lack of objective is justified by the fact that our approach provides the outline of the optimal path as input. In this context, our constraints provide an approximation of the robot's kinematic limitations, as well as the geometric limitations of the environment.

We employ the Gurobi Solver [13] in our implementation, as it is a common industry standard that is used in the state-of-the-art paper on which we base our formulation [31]. In the state-of-the-art, as well as the approach we introduce, it is understood that the valid walkable space is some collection of convex regions. In this paper, we refer to this collection of regions as the *environment*. In the following chapter, We provide a brief introduction to the concepts through which we can formally define an environment.

2.1 Introduction to Optimisation

2.1.1 Convexity

Convexity can exist in many forms [4]. We limit our definition to sets, as it is relevant to our formulation.

Definition 2.1.1. Convex set - A set $\mathcal{S} \subseteq \mathbb{R}^n$ is *convex* if and only if for any pair of

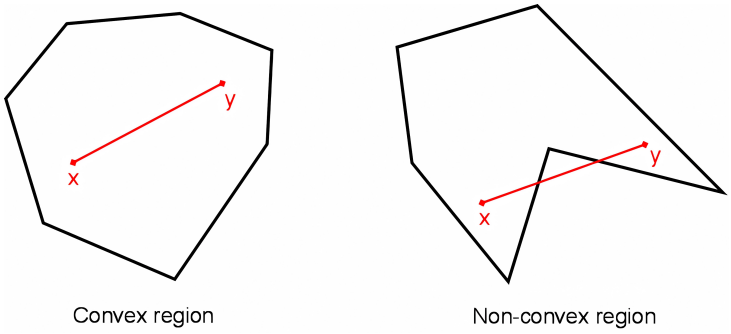


Figure 2.1: Two regions in $\mathbb{R}^2$. the left region is convex, and the right region is non-convex. [22]

points $\alpha_1, \alpha_2 \in \mathcal{S}$,

$$\lambda\alpha_1 + (1-\lambda)\alpha_2 \in \mathcal{S}$$

holds for all $\lambda \in [0,1]$.

Example 2.1.2. $\mathbb{R}^2$, the set of all real-valued two-dimensional vectors, is a convex set.

For our purposes, we will limit the notion of convexity to 2 dimensions, in that $\mathcal{S} \subseteq \mathbb{R}^2$. With this limitation, we can precisely define a set to be convex if, using α_1, α_2 as previously stated, every point on the line between α_1 and α_2 is contained within $\mathcal{S}$. It also allows us to visualise the concept easily such as in Figure 2.1.

2.1.2 Linear Constraints and Halfspaces

Definition 2.1.3. For $a_i, x_i, c \in \mathbb{R}$, a linear constraint is an equation of one of the following forms:

$$\sum_{i=1}^{n} a_i x_i = c, \quad (1)$$

$$\sum_{i=1}^{n} a_i x_i \geq c, \quad (2)$$

$$\sum_{i=1}^{n} a_i x_i \leq c, \quad (3)$$

where n is the number of dimensions of the input space, a_i for $i \in \{1..n\}$ are constant coefficients of the equation, x_i for $i \in \{1..n\}$ are our input variables, and c is our constraint value.

In short, in $\mathbb{R}^2$, a linear constraint of form (1) constrains the solution space to a single line. However, linear constraints of forms (2) and (3) will introduce a *halfspace* in which the solution must lie.

Definition 2.1.4. A halfspace is either of the two parts into which a hyperplane divides n-dimensional space [34]. Such a hyperplane is exactly that which is described by a linear constraint of form (1), whereas the entire space occupied by a halfspace is described by a linear constraint of the form (2) or (3).

A solution space defined by m linear constraints of forms (2) and (3) will be the intersection of m halfspaces. A value which lies in this space is known as a *feasible solution*.

We will continue to limit our scope to $n = 2$ for the remainder of this text, as it is the space in which our entire formulation and its solutions lie. A 2-dimensional halfspace is illustrated in Figure 2.2.

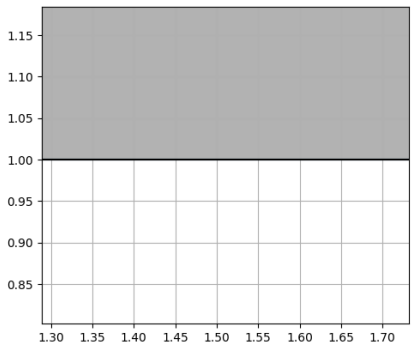


Figure 2.2: A 2D grid illustrating a linear constraint represented by a horizontal line at 1 on the vertical axis, dividing the space into two distinct halfspaces on the top and the bottom respectively.

2.1.3 Basics of Mixed Integer Programming

Mixed-Integer Programming (MIP) is a class of optimisation problems that involve both continuous and discrete variables. In this section, we provide an overview of the basics of MIP and its applications in various fields.

MIP problems are typically expressed as a set of linear or non-linear constraints, with a linear or non-linear objective function. The main thread between all MIP problems is the existence of integer decision variables. There are a number of classes and abbreviations for different circumstances. For instance, if all constraints are linear, we have a Mixed-Integer Linear Program (MILP). Otherwise, we have a Non-Linear Mixed Integer Program (NL MIP). In our case, we have a convex feasible region, linear objective function, a mixture of integer and real decision variables, and non-linear constraints. Hence, we have a Mixed-Integer Quadratically Constrained Program (MIQCP) [19]. This is what we will refer to as MIP throughout the text. A MIQCP problem can be expressed mathematically as:

$$\begin{aligned} \text{Minimise or Maximise} \quad & f(x) \\ \text{subject to:} \quad & x^T Q x + A_i^T x \leq c_i \quad \text{for } i \in \{1..m\}. \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^n$ is a vector of decision variables associated with linear and non-linear constraints, and some elements of x are constrained to be integers. $f : \mathbb{R}^n \to \mathbb{R}$ is a linear objective function, $A \in \mathbb{R}^{m \times n}$ is a matrix representing linear constraints, $Q \in$

$\mathbb{R}^{n\times n}$ is a symmetric matrix representing non-linear constraints such that $x^T Qx \geq 0$, and $c \in \mathbb{R}^m$.

MIP problems can be found in various fields such as agriculture, scheduling, logistics, and manufacturing [35]. For example, MIP has been applied to scheduling for rapeseed harvesting [6]. In logistics, the vehicle routing problem, where the goal is to minimie the total distance traveled by a fleet of vehicles, can also be formulated as a MIP problem [1].

The methods employed by solvers will not be elaborated upon, as they are outside the scope of this paper.

2.2 Convex Hull

Using our definition of convexity of sets, we can approach the idea of a *convex hull*, also called a *convex closure* or a *convex envelope*.

Let us define an arbitrary set χ of points in $\mathbb{R}^n$. The convex hull $\text{Conv}(\chi)$ of χ is the smallest convex set which contains every element of χ[1]. We will not elaborate on methods to compute convex hulls, as they are outside the scope of this paper.

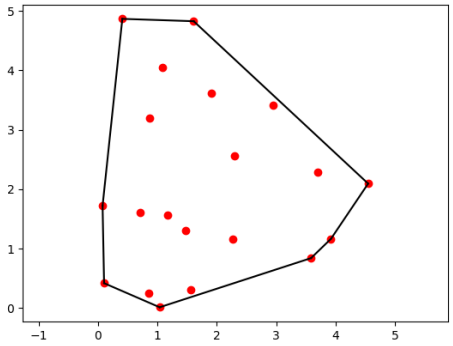


Figure 2.3: The convex hull of a set of points in 2 dimensions.

Recall that every convex hull is a *closed convex polygon*. An important formality to note is that of a *convex combination*. We understand that all points in a polygon can be expressed as $\sum_{i=1}^{n} \alpha_i v_i$ where n is the number of vertices, $\alpha_i \geq 0$ for $i \in \{1,\ldots,n\}$ such that $\sum_{i=1}^{n} \alpha_i = 1$ and $\{v_1,\ldots,v_n\}$ are the vertices of the polygon, if and only if the polygon is a convex hull. We introduce this extra information as a means to prove Proposition 4.1.2 later in the paper.

[1] When we say "smallest", we mean that $\text{Conv}(\chi)$ is a subset of or equal to all convex sets containing every element of χ.

Chapter 3

Formulation of Constraints on Contact Planning

3.1 Environment Constraints

In this formulation of the problem, the environment is comprised of a set of linear constraints of the form described in the previous chapter. Take, for instance, the constraint $x_2 \leq x_1$. We would write this in the form $[-1,1]^T[x_1,x_2] \leq 0$, which clearly shows our A, x, and c values.

Of course, no environment we come across in real life will be described entirely by this single constraint, or any single constraint for that matter, as the feasible region would be the infinite region defined by a single halfspace such as in Figure 2.2. We need to add at least two more constraints to construct a closed polygon as in Section 2.2, as we are working in two dimensions [21]. Let us introduce the constraints $x_1 + x_2 \leq 1$ and $x_2 \geq 0$. These constraints are formulated as $[1,1]^T[x_1,x_2] \leq 1$ and $[0,-1]^T[x_1,x_2] \leq 0$ respectively.

Formulating a linear program of all of these constraints is trivial due to the way we have structured each of them. It is a simple case of "stacking" each respective A and c to obtain a vectorised inequality. In this case we have

$$\begin{bmatrix} -1 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix}^T [x_1,x_2] \leq \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

The resulting constrained region in which x must lie is in the shape of a triangle and is illustrated in Figure 3.1.

However, the vast majority of the environments we wish to deal with have a lot more than one region (see Figure 3.3). To add another region, let us introduce three constraints which form another triangle which is completely disjoint from our first triangle.

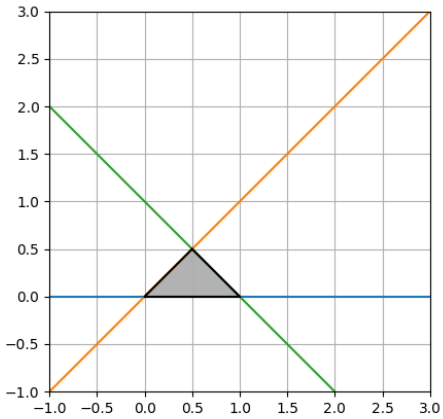


Figure 3.1: A triangle formed by intersecting three halfspaces in 2D space.

Our resulting $A^T x \leq c$ formulation will be as follows:

$$\begin{bmatrix} -1 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix}^T [x_1, x_2] \leq \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ OR } \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & -1 \end{bmatrix} [x_1, x_2] \leq \begin{bmatrix} 4 \\ -1 \\ -1 \end{bmatrix}$$

We can group each region's constraints together, giving us a neater notation. Rewriting the above inequality, we obtain:

$$A_1^T x \leq c_1 \quad \text{OR} \quad A_2^T x \leq c_2.$$

But this is not a valid MIP. To convert this into an equivalent MIP, we simulate the OR constraint using a method known as the *big-M formulation* [12].

To this end, we introduce integer variables to describe exactly which set of constraints to which we wish to adhere. More specifically, we will introduce a vector of binary variables b which describes which constraints we must adhere to using the *big M formulation*. For instance, if we only wish to adhere to the set of constraints that describes the region in figure (whatever the first triangle was). Our formulation will look as follows:

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}^T [x_1, x_2] \leq \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} * M.$$

Where M is a scalar that is sufficiently large i.e 1000. This addition, as we can see, nullifies the effect of the second set of constraints by making their c values sufficiently high such that any reasonable value of x would satisfy $A_2^T x \leq c_2$. Any value of M that achieves this is sufficiently large.

In the future, for the sake of clarity and intuition, we will add $(1-b) * M$ instead of $b * M$ so that we can treat constraints with corresponding binary variable 1 as "in use" and to disregard those with 0.

Finally, we obtain:

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}^T [x_1, x_2] \leq \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \left(1 - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) * M,$$

which further simplifies to something close to the linear part of our established form in 2.1.

$$A^T x \leq c + (1 - b) * M.$$

Finally, we must constrain the binary variables to allow *exactly one* set of constraints to be "in use" for any given contact point.

To do this, we add the constraint $\sum_s^n b_s = 1$.

3.2 Model Constraints

We aim to construct a reachable region similar to the regions used in state-of-the-art applications. This region usually consists of some enforced reachable distance, as well as some extra positional considerations to approximate kinematic limitations [29], [37].

3.2.1 Reachability Constraints

The dynamic elements of robot locomotion are coarsely approximated for this formulation. This approximation manifests as a constant distance constraint between each contact point. We will define it as follows:

Let $x = \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ \vdots & \vdots \\ x_{n,1} & x_{n,2} \end{bmatrix}$ be our vector of 2-dimensional contact positions for n steps.

For all $s \in \{1..n-1\}$, let x be subject to:

$$\sqrt{(x_{s,1} - x_{(s+1),1})^2 + (x_{s,2} - x_{(s+1),2})^2} \leq r$$

For some $r \in \mathbb{R}$ that defines the reachable distance of the robot in one step. It is understood that this approximation coarsely models the dynamics of robot locomotion, and more accurate models can be used depending on the specific application. In simple terms, the reachability constraint simply constrains consecutive contact points to be within a certain distance r of each other.

3.2.2 Positional Constraints

This formulation of the problem does not take the angle of the foot at each contact point into account. Therefore, our robot will always face the same direction (this direction is "up" on the visualisations). With this assumption, we wish to simulate bipedality.

Our assumption allows us to add a simple constraint to each step. This constraint dictates that the robot's left foot cannot be farther to the right than its right foot and vice versa i.e the robot's legs cannot "cross over". In addition to this, we add a minimum separation requirement between the feet, we call this the "offset" and notate it as Δ. We introduce this constraint as an approximation of the limitation of joint angles of the robot [37]. We suggest Δ to be at least the width of the robot's foot.

This is enforced by the following sets of linear constraints:

For all s $\in \{1..n-1\}$ such that $s \mod 2 = 0$, let x be subject to:

$$x_{s,1} \leq x_{(s+1),1} - \Delta.$$

And for s such that $s \mod 2 = 1$:

$$x_{s,1} \geq x_{(s+1),1} + \Delta.$$

Where $\Delta > 0$ is the minimum horizontal separation between contact points.

If we use these constraints, we acquire $x_{1,1} \geq x_{2,1} + \Delta$. In other words, the first contact point must be more to the "righ" than the second contact point. For this to make sense, we must set our first contact point $(s = 1)$ as being achieved with the right foot, and the second contact point $(s = 2)$ being achieved with the left foot, with steps alternating throughout the contact plan. In order to change the starting foot, we may flip the 1 and 0 modulo constraints. Recall that this assumption is only reasonable when we make the simplification in which the robot always facing directly in the vertical axis.

It is important to note that, although these ideas have been developed independently and without reference for this dissertation, similar ideas can be found to exist in literature [37] [29], such as in Figure 3.2.

3.2.3 Start and End Point Constraints

The robot's first and final contact positions are fixed in this formulation of the problem. There do exist some formulations in which the start and end points are merely suggested, with an included objective function which minimises the first and last contact points' distances from the provided start and end points respectively [31]. However, in this formulation, we simply fix the first and last footstep positions with the following constraints.

Let $x_\alpha, x_e \in \mathbb{R}^2$ be our desired start and end points respectively, let x be subject to:

$$x_1 = x_\alpha,$$
$$x_n = x_e.$$

Note that we could instead set start and end *regions* by introducing constraints on the integer variables corresponding to $s = 1$ and $s = n$ respectively.

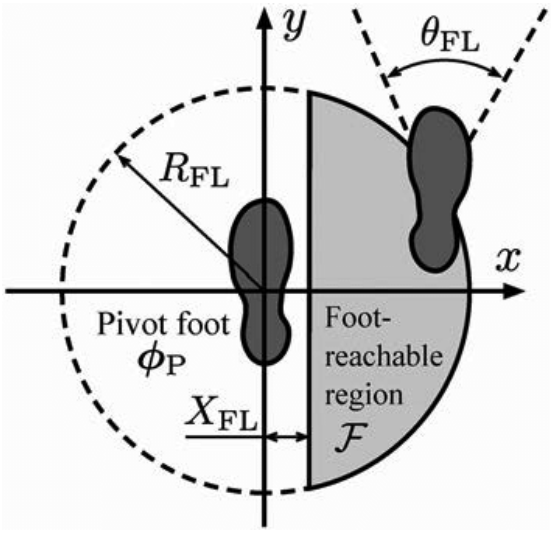


Figure 3.2: Reachable region adopted in prior work [37]. The value we notate Δ is notated X_{FL} and the value we notate r is notated R_{FL} in this work.

3.3 The Full MIP

Combining each of these constraints into a formal MIP formulation, our result is the following:

Let n be the number of steps taken.

Let r be the maximum distance between each step.

Let S be the set $\{1,..,n\}$

Let A, c be the coefficients of the linear equations describing all of the regions of the environment.

Let M be sufficiently large as to dominate all of the terms in c.

Let $\Delta > 0$ be the minimum horizontal separation between footsteps.

Let x_α, x_e be the start and end points of the environment respectively.

For each $s \in S$, find x_s and b_s,

$$\begin{aligned}
\text{subject to:}\quad & A^T x_s \leq c + (1 - b_s) * M, \\
& \sqrt{(x_{s1} - x_{(s+1)1})^2 + (x_{s2} - x_{(s+1)2})^2} \leq r, \\
& \sum b_s = 1, \\
& x_s = x_\alpha && \text{if } s = 1, \\
& x_s = x_e && \text{if } s = n, \\
& x_{s,1} \leq x_{(s+1),1} - \Delta && \text{if } s \bmod 2 = 0, \\
& x_{s,1} \geq x_{(s+1),1} + \Delta && \text{if } s \bmod 2 = 1.
\end{aligned} \tag{3.1}$$

Where b_s is a vector of binary variables denoting whether or not each set of linear

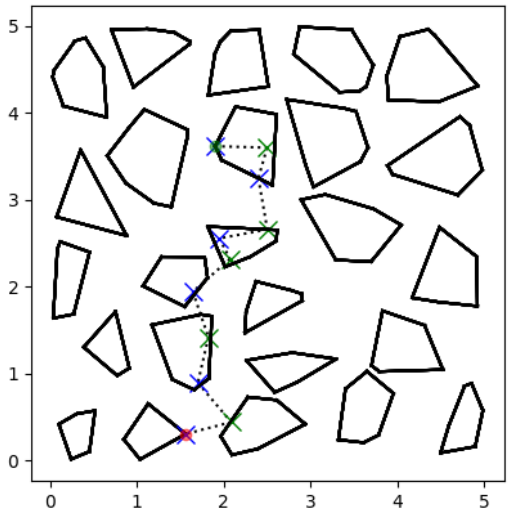


Figure 3.3: A solution, generated by Gurobi, of the MIP we propose in Section 3.3 over an environment randomly generated using K-means clustering. Contact points with the left foot are marked when a blue $\times$, and right with a green $\times$. The end point is marked with a red point and the start point is marked with a green point. $r = 0.6, n = 11, \Delta = 0.1$.

constraints is "in use" for step s, x_s is the position of step s on the (x_1, x_2) axis.

The output of this MIP is a contact plan of length n satisfying these constraints, an example is shown in Figure 3.3.

3.4 Combinatoric Consequences and Heuristic Improvements.

Given this model, we can analyse the number of possible permutations of our vector b of binary variables to place an upper bound on the number of discrete values we must fix in order to find a solution to the problem. If m is the number of walkable regions, and n is the number of steps taken, we can see that the upper bound on the number of combinations of walked regions is m^n. This may also be thought of as the number of permutations of an $n \times m$ matrix B with $B_{ij} \in \{0,1\}$ for $i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}$, where each column B^i of length m denotes the region in which step i is located. Of course, with an advanced solver such as Gurobi, this upper bound is only theoretical as there exist "pre-solving" procedures which can transform the model into an equivalent model of greatly reduced complexity before attempting to solve it [17].

Our notion of reachability as it stands is already formed heuristically, due to the fact that dynamics are not being considered. In some cases, the combinatoric complexity of the problem may be reduced through additional heuristic means. In the state-of-the-art, these improvements involve rough filtering through domain-specific heuristics

which can vary from environment to environment [11]. These changes may reduce the combinatoric complexity by systematically eliminating certain parts of solution space as an option for certain steps.

Note that, due to our simplification of the problem, we have an exact analytical representation of the reachable space from any given point. In the following chapter, we will leverage this to provide an exact representation of the total reachable space in n steps for any environment. Using this representation, we construct a graph which reduces the complexity of the initial combinatoric problem to zero by providing an exact matrix B which is guaranteed[1] to be the minimum possible size and therefore provide a plan consisting of the lowest possible number of steps. Additionally, the way we construct the graph allows us to compute alternative contact plans more quickly than the MIP method we have introduced so far. In the following section, we elaborate on why these fast alternative plans may be useful.

[1]Depending on how high n is for the linearisation of the reachable space, see Section 4.1.1

Chapter 4

Breadth-first Graph Construction Approach

In the following chapter, we define a graph for representing the reachable space in an environment. In the graph, the root node represents some point in the environment, and any node with n degrees of separation from the root node represents a continuous space reachable[1] by the robot in n steps from the root. Each node is logically connected to the node from which it was reached, inspiring a tree where the depth of a node corresponds to its step number. As each node represents a part of some region within the environment, we can draw a direct parallel between traversal of the graph and traversal of the environment. Specifically, a path through the graph is exactly the sequence of regions which allow us to reach the region represented by the node at the end that path in the graph. This sequence of regions is then directly translated to a matrix B containing the vector b_s for each step as its columns. The process through which we compute the graph will now be introduced.

4.1 Walkable Region Acquisition

In order to compute the children of a node which represents a region, we must introduce a way to compute the reachable space from any given region. Recall that we already have the exact reachable region from any point ϕ in our environment, namely $\mathcal{P}_\phi$. This section extends our notion of reachability and introduces a way to "apply" $\mathcal{P}$ to a *region* instead of a point, thereby computing the reachable region from that region, allowing us to generate its children in the graph we aim to produce.

4.1.1 Linearisation of Quadratic Reachability Constraints

We are motivated to provide a linearised version of our reachability constraint in order to apply Proposition 4.1.2 to the reachable region we acquire when adhering to the reachability constraint from a point. We will construct our linearised region with linear constraints acquired by computing $n \in \mathbb{N} : n \geq 2$ evenly spaced points on a truncated

[1] According to the constraints laid out in Section 3.3.

circle. We will define a vector $L \in \mathbb{R}^{2\times n}$ of points on the truncated circle. The reachable region will be different depending on the foot that is taking the step, so we introduce the exponents $L^{\mathbf{L}}, L^{\mathbf{R}}$ to denote the the points defining the left and right foot's linearised reachable region respectively.

Let r be our reachability constraint as introduced in Section 3.2.1. Let Δ be the offset in our positional constraints as introduced in Section 3.2.2. Let

$$\Theta_\Delta = \sin^{-1}\left(\frac{\Delta}{\sqrt{r^2+\Delta^2}}\right).$$

For $m \in \{0..n\}$, we let:

$$L^{\mathbf{L}}_m = \left(\left(r\cos\left(\left(\frac{\pi}{2}+\Theta_\Delta\right)+\frac{m}{n}(\pi-2\Theta_\Delta)\right)\right), \left(r\sin\left(\left(\frac{\pi}{2}+\Theta_\Delta\right)+\frac{m}{n}(\pi-2\Theta_\Delta)\right)\right)\right),$$

$$L^{\mathbf{R}}_m = \left(\left(r\cos\left(\left(\frac{3\pi}{2}+\Theta_\Delta\right)+\frac{m}{n}(\pi-2\Theta_\Delta)\right)\right), \left(r\sin\left(\left(\frac{3\pi}{2}+\Theta_\Delta\right)+\frac{m}{n}(\pi-2\Theta_\Delta)\right)\right)\right).$$

For $\mathcal{F} \in \{\mathbf{L}, \mathbf{R}\}$, we define the canonical polytope, notated $\mathcal{P}_0^{\mathcal{F}}$, as the convex hull taken over the resultant points $L^{\mathcal{F}}$ of the truncated circle. In other words, $\mathcal{P}_0^{\mathcal{F}} = \text{Conv}(L^{\mathcal{F}})$. This is illustrated in Figure 4.1d. Note that this linearisation provides an inner approximation, in that all points in the linearised region are guaranteed to be contained within the actual walkable region. Also note that as we increase n, our approximation gets more accurate to the real region, with theoretical equality at $n = \infty$.

Now, we can see that the reachable region from any point $\phi \in \mathbb{R}^2$ may be calculated as a simple translation of $\mathcal{P}_0^{\mathcal{F}}$.

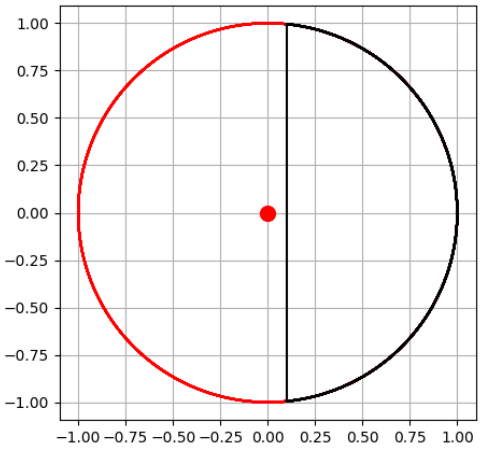
Lemma 4.1.1. *Let $\mathcal{F}', \mathcal{F} \in \{\mathbf{L}, \mathbf{R}\}$ such that $\mathcal{F}' \neq \mathcal{F}$. Let $\phi \in \mathbb{R}^2$ be a contact point of foot $\mathcal{F}'$. The reachable region from ϕ will be notated as $\mathcal{P}_\phi^{\mathcal{F}}$ such that $\mathcal{P}_\phi^{\mathcal{F}} = \{x+\phi : x \in \mathcal{P}_0^{\mathcal{F}}\}$. For a point $x \in \mathcal{P}_0^{\mathcal{F}}$, we say that point $(x+\phi)$ is* equivalent to x with respect to $\mathcal{P}_\phi^{\mathcal{F}}$.

In the future, we will refer to $\mathcal{P}_\phi^{\mathcal{F}}$ as $\mathcal{P}_\phi$ when specifics about the foot are not relevant. Keep in mind that, with repeated use of $\mathcal{P}_\phi$ in the same section, the foot is understood as staying the same over all mentions of $\mathcal{P}_\phi$ unless stated otherwise.

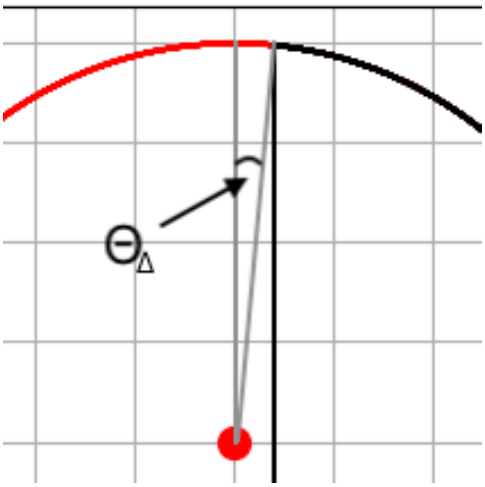
4.1.2 Reachability From a Region as Opposed To a Point

So far, our notion of reachability has been with respect to a single point. In this section, we will extend this notion to the entire reachable area from a region of space. We are motivated to calculate this in order to find the children of a node in our graph, as all nodes in the graph except the root are represented as regions, not points.

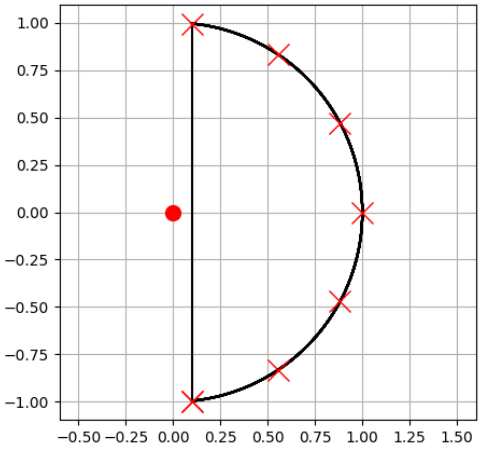
Proposition 4.1.2. *Let $\mathcal{R}$ be a non-empty, 2-dimensional, closed convex polygon. Let $\mathcal{P}_\phi$ be a polytope of the form established in Section 4.1.1 with starting point ϕ such that $\phi \in \mathcal{R}$. Let $\{v_1, v_2, \ldots, v_n\}$ be the vertices of $\mathcal{R}$. The total reachable space from any possible point in $\mathcal{R}$, $\bigcup_{\phi\in\mathcal{R}} \mathcal{P}_\phi$, notated $\mathcal{P}(\mathcal{R})$, is exactly equal to Conv$(\bigcup_{i=1}^n \mathcal{P}_{v_i})$*



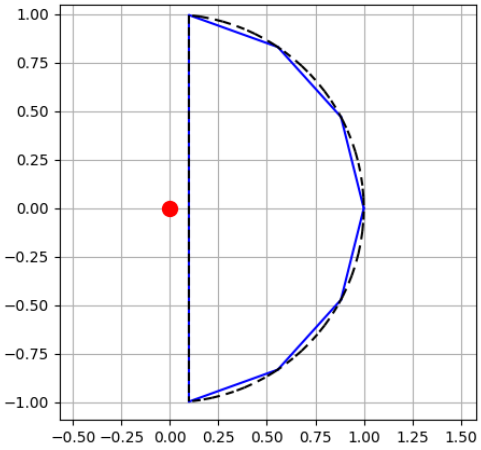
(a) The actual reachable region of the right foot in black, with a red circle around the origin for reference. The position of the left foot is marked as a red point.



(b) A close-up of 4.1a, highlighting the angle Θ_Δ.



(c) Points on the reachable region of the right foot ($L^{\mathbf{R}}$), with $L^{\mathbf{R}}_m$ marked by a red $\times$ for each $m \in \{0, \ldots, n\}$.



(d) The linearised reachable region of the right foot in blue, created by computing Conv($L^{\mathbf{R}}$).

Figure 4.1: A sequence of figures illustrating the construction of the linearised reachable region $\mathcal{P}^{\mathbf{R}}_0$, such that $r = 1, \Delta = 0.1, n = 6$, with the origin marked as a red dot.

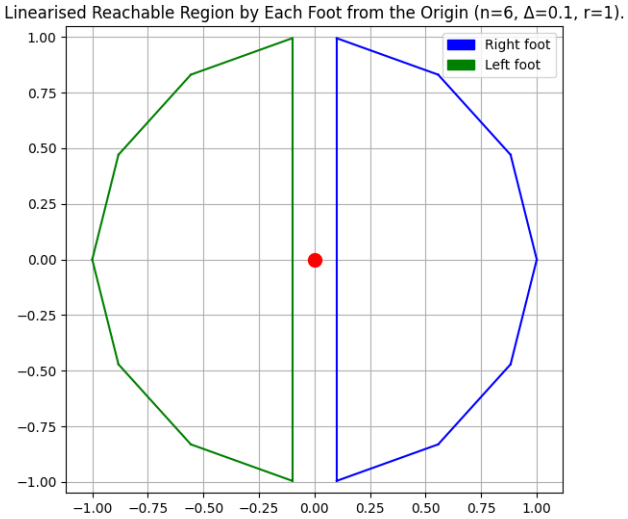
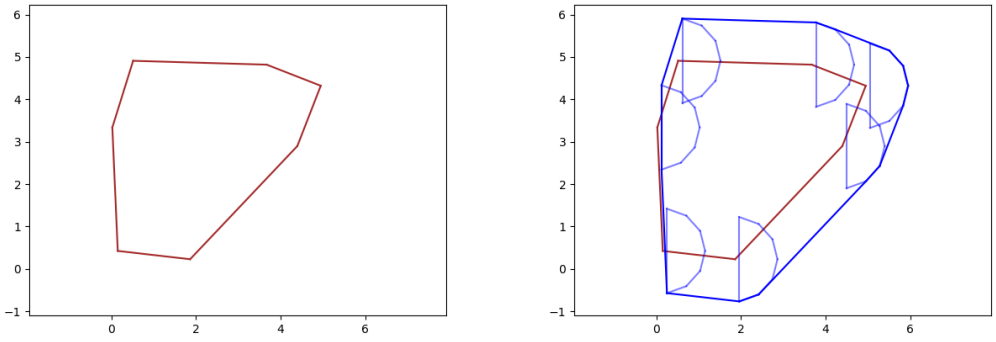


Figure 4.2: A visualisation of $\mathcal{P}_0^{\mathbf{L}}$ (green region) and $\mathcal{P}_0^{\mathbf{R}}$ (blue region), the linearised reachable region from the origin (marked with a red dot) for the left and right foot respectively such that $n = 6, \Delta = 0.1, r = 1$.



(a) A convex region.

(b) The region reachable by the right foot from region 4.3a, constructed by taking the convex hull of the polytope constructed at each of its vertices.

Figure 4.3: A convex region, next to an illustration of its reachable region being computed according to Proposition 4.1.2.

Proof. Take an arbitrary point $x \in \bigcup_{\phi \in \mathcal{R}} \mathcal{P}_\phi$. By definition, there exists some $\phi \in \mathcal{R}$ such that $x \in \mathcal{P}_\phi$. Since $\mathcal{P}_\phi = \{x + \phi : x \in \mathcal{P}_0\}$, we can write $x = y + \phi$ for some $y \in \mathcal{P}_0$.

We show that ϕ is a convex combination of the vertices of $\mathcal{R}$, i.e. $\phi = \sum_{i=1}^{n} \alpha_i v_i$ for $\alpha_i \geq 0$ with $\sum_{i=1}^{n} \alpha_i = 1$, and that $\sum_{i=1}^{n} \alpha_i y = y$. Therefore,

$$x = y + \phi = y + \sum_{i=1}^{n} \alpha_i v_i = \sum_{i=1}^{n} \alpha_i y + \sum_{i=1}^{n} \alpha_i v_i = \sum_{i=1}^{n} \alpha_i (y + v_i).$$

Since $y \in \mathcal{P}_0$, the translated point $y + v_i \in \mathcal{P}_{v_i}$ for all $i \in \{1, \ldots, n\}$. Thus, x is a convex combination of points in $\bigcup_{i=1}^{n} \mathcal{P}_{v_i}$, and hence $x \in \text{Conv}(\bigcup_{i=1}^{n} \mathcal{P}_{v_i})$. Thus we have shown that $\bigcup_{\phi \in \mathcal{R}} \mathcal{P}_\phi \subseteq \text{Conv}(\bigcup_{i=1}^{n} \mathcal{P}_{v_i})$.

Now, for the other direction, take an arbitrary point $x \in \text{Conv}(\bigcup_{i=1}^{n} \mathcal{P}_{v_i})$. By definition, this means x can be written as a convex combination of points from $\bigcup_{i=1}^{n} \mathcal{P}_{v_i}$, i.e. $x = \sum_{i=1}^{n} \beta_i (y_i + v_i)$ for $\beta_i \geq 0$ with $\sum_{i=1}^{n} \beta_i = 1$ and points $y_i \in \mathcal{P}_0$.

Define $\phi = \sum_{i=1}^{n} \beta_i v_i$. Since $\mathcal{R}$ is a closed convex polygon, $\phi \in \mathcal{R}$. Let $y = \sum_{i=1}^{n} \beta_i y_i$. Since $y_i \in \mathcal{P}_0$ and convex combinations preserve convexity, y is also in $\mathcal{P}_0$. Thus, we can express x as the following:

$$x = \sum_{i=1}^{n} \beta_i (y_i + v_i) = \sum_{i=1}^{n} \beta_i y_i + \sum_{i=1}^{n} \beta_i v_i = y + \phi$$

Since $\phi \in \mathcal{R}$ and $y \in \mathcal{P}_0$, it follows that $x \in \mathcal{P}_\phi$. This implies that $x \in \bigcup_{\phi \in \mathcal{R}} \mathcal{P}_\phi$. Therefore, we have shown that $\text{Conv}(\bigcup_{i=1}^{n} \mathcal{P}_{v_i}) \subseteq \bigcup_{\phi \in \mathcal{R}} \mathcal{P}_\phi$.

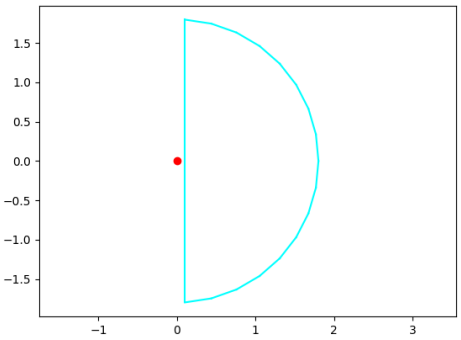
We conclude that $\bigcup_{\phi \in \mathcal{R}} \mathcal{P}_\phi = \text{Conv}(\bigcup_{i=1}^{n} \mathcal{P}_{v_i})$. □

In other words, the reachable region from a region is the convex hull of the reachable regions from its vertices, as shown in Figure 4.3. Naturally, this brings us back to the reason we introduced the linearisation of the reachable region. We can apply this proposition to any region with a finite number of vertices, so we find the reachable region in n steps from some point, provided that all space is walkable space in Figure 4.4.

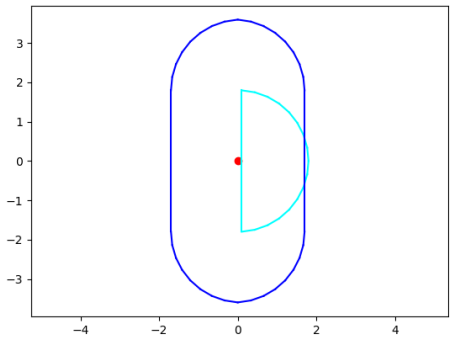
4.2 Graph Construction

Recall that an *optimal path* between two points is one which involves the minimum possible number of steps.

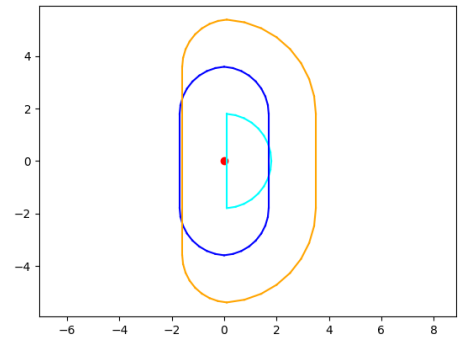
We begin by considering a tree. One powerful property of trees we are interested in is their innate encoding of a path from any node in the tree to the tree's root node as illustrated in Figure 4.6. Going forward, we will use the term "up-traversal" to refer to the traversal towards the root of a tree from any node in that tree by looking up the parent of our node, then the parent of the parent and so on, until we reach the root. Note that this procedure is faster than any possible traversal algorithm, in that it involves no unused computation and takes exactly the number of iterations as the depth of the tree.



(a) In cyan, the reachable region $\mathcal{P}_0^{\mathbf{R}}$ from the origin in 1 step with the right foot.



(b) In blue, the reachable region $\mathcal{P}^{\mathbf{L}}(\mathcal{P}_0^{\mathbf{R}})$ from the origin in 2 steps.



(c) In orange, the reachable region $\mathcal{P}^{\mathbf{R}}(\mathcal{P}^{\mathbf{L}}(\mathcal{P}_0^{\mathbf{R}}))$ from the origin in 3 steps.

(d) A colour-coded graph representing these three steps taken in a single continuous region. If there is only one continuous region, each node will always have at most 1 child.

Figure 4.4: An illustration of the expansion of reachable space as the number of steps increases in a single continuous region. $(n = 16, \Delta = 0.1, r = 1.8)$. Observe how this demonstrates that, in an environment with no discontinuities, the graph does not branch. By extension, see that branching of the graph represents discontinuities in the environment.

In this context, when we say a path is "encoded" in our tree, it means that we can find the path using up-traversal.

Using the concepts introduced so far, we will provide techniques for creating a useful graph of the environment as described above which, in addition to providing a quickly computable optimal path between our desired start and end points, provides an equally quickly computable optimal path from any region within a range of its root node to its root node. Additionally, some of these techniques allow us to compute alternative, or even all paths to some desired point from any point in the environment, as long as the path is shorter than the optimal path between the start and end point (given that we terminate once we find the path between the start and end). These less-than-optimal paths will prove useful in circumstances we introduce in 4.3.3. The main drawback of this approach is the time it takes to produce the graph offline. The differences in the following construction approaches lie in the pruning of the graph in the interest of efficiency. It is clear to see that computation of every theoretical node of the graph is very expensive, as we will see, hence we introduce methods to proactively reduce the size of the graph as we construct it in Section 4.2.

The choice of root node will influence greatly the type of paths we encode. For instance, if we let our starting point (3.2.3) be the root node, we encode the path from the start point to any region corresponding to a node in the tree. However, if we let the end point be the root node, we encode every path from any region in the tree to the end point. Both of these methods encode the optimal path between the given start and end point, but we will prefer the end point as our root node. This is chosen as it addresses the problem of uncertainty in the position of the robot, allowing it to re-plan in the event where it makes some step in a region that is not part of the calculated optimal path. We explore this idea in Section 4.3.

Note that the way we explore the environment from the end to the start is assumed to work in reverse to find a path from the start to end. This fact relies on a few key assumptions. Namely, the polytope representing reachable region with foot $\mathcal{F}$ from a point p must also be the polytope representing all points such that p can be reached with foot $\mathcal{F}'$. Formally, we must have the equality

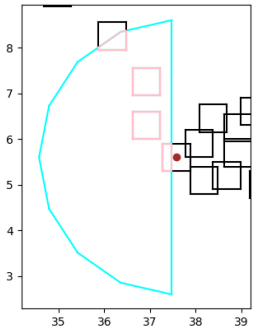
$$\mathcal{P}_{\phi}^{\mathcal{F}} = \{p : \phi \in \mathcal{P}_{p}^{\mathcal{F}'}\}.$$

This is satisfied in our model by the fact that our approximation of the reachable region is always one of two polytopes as described in section 4.1.1 which satisfy this property due to the quasi-static assumption, and the limitation of always facing "up".

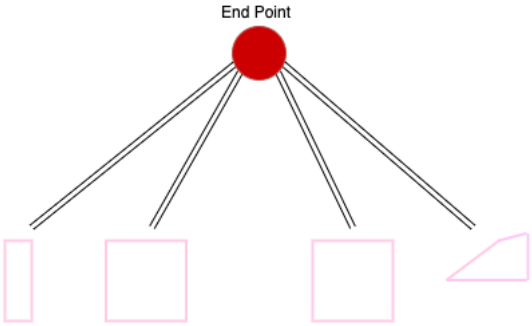
We let $\mathcal{F}, \mathcal{F}' \in \{\mathbf{L}, \mathbf{R}\}$, such that $\mathcal{F} \neq \mathcal{F}'$. We introduce this notation to abstract away from specific left and right foot behaviour, and to only be concerned with the alternation of feet.

4.2.1 Full Tree

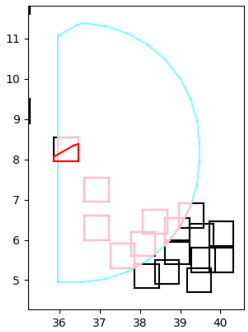
Let $\mathcal{R}$ be an environment as described in Section 3.1. As we have established, we begin at the *end point* of $\mathcal{R}$, denoted (x_e) as described in 3.2.3. We let (x_e) be our root node. To compute its children, we begin by computing $\mathcal{P}_{x_e}^{\mathcal{F}}$ as described in Section



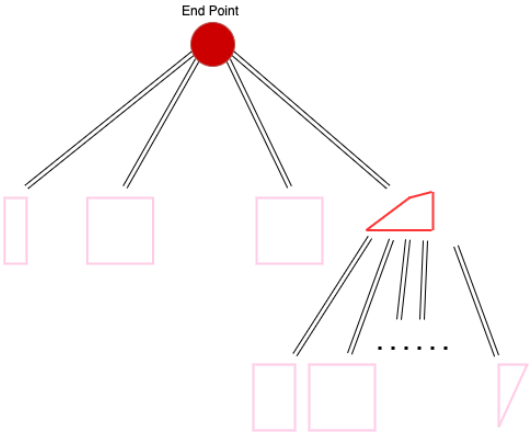
(a) The intersection of $\mathcal{P}^{\mathbf{L}}_{x_e}$ with the environment.



(b) x_e and its children, illustrated as a graph. Let us denote the right-most node in this image N_1.



(c) $\mathcal{P}^{\mathbf{R}}(N_1)$, in cyan. The intersection of $\mathcal{P}^{\mathbf{R}}(N_1)$ with the environment is in pink.



(d) The graph after computing the children of N_1.

Figure 4.5: *The graph construction in action* - (a) An illustration of $\mathcal{P}^{\mathbf{L}}_{x_e}$, the reachable space by the left foot, in cyan, from the end point (x_e) of some environment. Four disjoint regions, in pink, are produced by the reachable region's intersection with the environment. These pink regions are the children of x_e, as they are reachable from x_e in one step. Note that each of these regions have been reached with the left foot, associating these nodes with the left foot. The same process is repeated for one of the children of x_e, associating each of its children with the right foot.

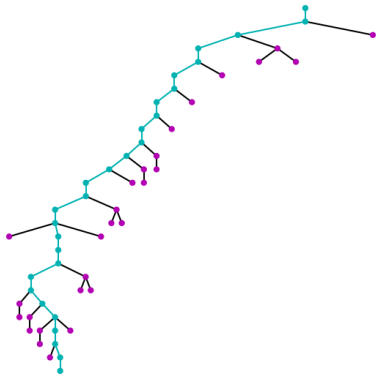


Figure 4.6: A tree, with a path between two nodes highlighted. Note that this path can be computed using up-traversal as it has an end at the root.

	Full Tree	Reduced Tree	Depth-aware Graph
Unexpected position (in graph)	✓✓✓	✓✓✓	✓✓✓
Unexpected position (not in graph)	✓✓✓[2]	✓✓	✓✓
Objective change	✓✓	✓	✓✓
Environment change	✓✓✓		✓✓✓
Number of nodes (50 steps 50 regions)	$\sim 10^{82}$	$\sim 10^{2}$	$\sim 10^{5}$
Number of nodes (10 steps 10 regions)	$\sim 10^{10}$	$\sim 10^{1}$	$\sim 10^{3}$

[2] No possible positions are outside the full graph.

Table 4.1: A summary of tradeoffs between different graph construction approaches in various online scenarios. The values for complexity of the Full Tree are theoretical, whereas the values for the Reduced Tree and Depth-aware Graph are simplifications of examples in practice. ✓✓✓: Able to construct an optimal path using up-traversal. ✓✓: Able to construct an optimal path without needing to create the entire graph again. ✓: Able to construct a path, but it is not necessarily optimal.

4.1.1. The intersection of $\mathcal{P}_{x_e}^{\mathcal{F}}$ with our environment produces $1 \leq k_{x_e} \leq m$ disjoint, convex regions as demonstrated in 4.5a. We let each of these regions be a child of (x_e). The rest of our construction follows similarly. For each node N represented by region R , its children are the $1 \leq k_N \leq m$ disjoint, convex regions acquired by intersecting the reachable region $\mathcal{P}^{\mathcal{F}'}(R)$ with the environment. Once we reach a step at which all children generated are already included in the tree, we terminate. Note that this is the specific case where we wish to construct the graph such that the end point of the environment is the root node, and we may also optionally terminate once a node containing x_α is found.

We say that a node produced by the intersection of some $\mathcal{P}^{\mathcal{F}}$ is *associated with* $\mathcal{F}$, and similarly for $\mathcal{F}'$. This will be important for re-planning, when trying to map the robot's new position to a node in the graph.

Note that this approach, at high number of steps and high number of regions in the environment, creates a tree that is not of feasible size to work with computationally (see Table 4.1). Hence, we introduce techniques which greatly reduce the branching factor by pruning redundant nodes. This pruning results in a decrease of theoretical capabilities of the graph, as again shown in Table 4.1.

4.2.2 Reduced Tree

This method attempts to skip encoding any paths that are not optimal. It does this by checking for inclusion. Before a node is added to the tree, a check is performed to check if that node is a subset of the union of all existing nodes. If it is a subset, we do not add the node to the tree, as its entire reachable region will already be included, meaning all of its potential children will be accounted for. Note that, since the larger region is already in the tree, it must be at a lower depth than the new node and therefore a encode a path with a lower number of steps.

The method to generate this tree is similar to that of the **Full Tree**, but with the above intermediate step included before the addition of each child of a node.

4.2.3 Depth-aware Graph

If we wish to preserve some alternative paths without explicitly encoding every single one of them, we can take a similar approach as before. Let N be an existing node, let c be a node that is not yet included in the graph and is a subset of N. This time, instead of simply not adding c, we add c to the graph as a child of its parent and additionally add an edge between c and the parent of N. Finally, do not compute the children of c on any of the following steps. Of course, this creates a loop and makes the graph technically no longer a tree. (We can see that there is at least one loop by observing the fact that there is now more than one path from c to the root). However, we can still use a form of up-traversal by taking the parent with the lowest depth as the next step in the path. We introduce this approach as a primitive extension to our approach which provides a more feasible approach to *changing environment* problems compared to a **Full Tree**. We discuss this in Section 4.3.3.

This approach does not prune as many nodes as the **Reduced Tree**, as this method must check for subsets of nodes as opposed to inclusion within the union of all existing nodes. This change makes a substantial difference, and makes this method very difficult to work with in that it takes a much longer time to construct (see Table 4.1). An improvement on this work would be to find some way – in the spirit of this approach – to represent a node that is not fully inside any single node, but is included in the union of nodes without adding an entirely new node. Currently, this limitation greatly hampers the performance of this method, meaning it will take multiple hours to construct a single graph for a complex environment with the implementation provided for this paper.

4.3 Re-planning

4.3.1 In the Event of an Unexpected Position

When a contact occurs in a region that is not the expected next region provided by the graph traversal, we must create a new plan. There are two scenarios which we must consider with respect to an unexpected position:

i. The unexpected position is already represented as a node in the graph, in which case we can simply acquire a new path through up-traversal from that node. See Figure 4.8.

ii. The unexpected position is not represented as a node in the graph. In this case, we provide a procedure which allows us to find a way to "re-connect" with the graph and run procedure (i) from the nearest node. See Figure 4.10.

In scenario (ii), the aforementioned procedure is as follows.

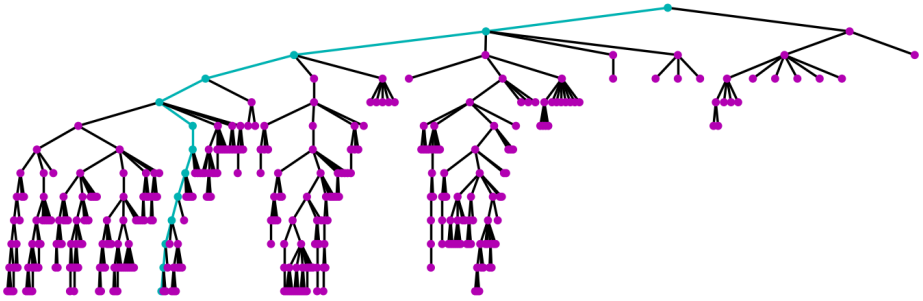
1. Create a new graph with root node at the unexpected contact point (x_u), associated with the foot which produced the contact.
2. Use one of the techniques in Section 4.2 to begin constructing a new graph which encodes all of the paths reachable from the unexpected position.
3. Compute levels of the new graph until some node in the new graph N_a intersects one some node N_b of the originally constructed graph (note that the intersection must be between two nodes associated with the same foot).
4. Use up-traversal to compute the route from x_u to N_a.
5. Similarly compute the route from N_b to x_e.
6. Concatenate the two sequences of regions, merging the last element of the first with the first element of the second i.e $[2,4,3],[3,5,9]$ becomes $[2,4,3,5,9]$.

Note that this generates the optimal path back into the graph, and therefore an optimal path to the end from the unexpected position.

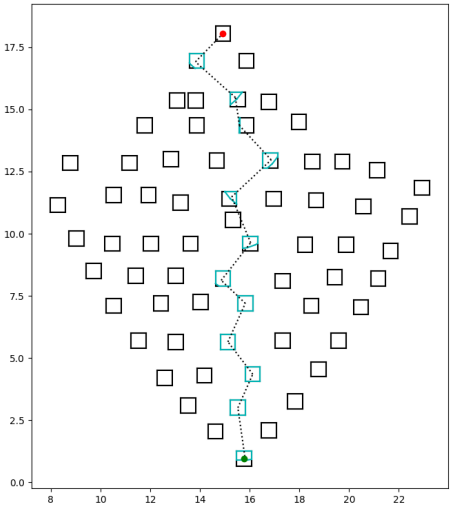
We assume that scenario i is more likely in the event of an unexpected position, as we assume that a disruption in the robot's locomotion is most likely caused by the sum of small uncertainties and inaccuracies in the kinematic model of the robot, or by some force manipulating the robot in an unexpected way. Furthermore, if the robot unexpectedly ends up in a region which is a single step's distance away from some region in its original plan, the unexpected region is likely to have been explored by the nature of the graph construction algorithm.

Note that Procedure ii generates paths which are only optimal under the constraint that the final footstep is made by the foot associated with the root node of the original graph. This is shown when, during implementation, there sometimes exist paths from the unexpected point to the end point with a lower number of steps than the path generated by Procedure ii. This is a minor limitation of this approach, and it would require the construction of 2 graphs, each starting with different feet, to completely solve this problem. The problem of starting on the most optimal foot is one which permeates

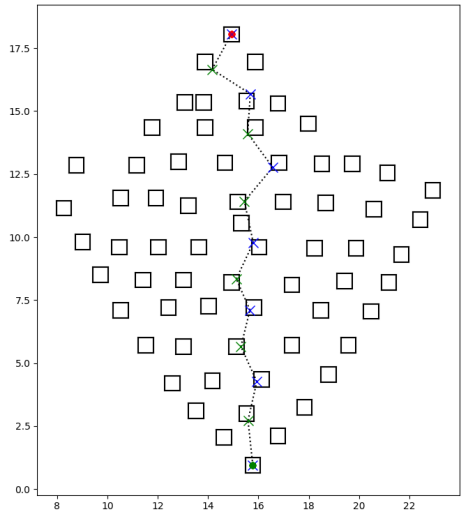
throughout the entire MIP approach, as it is another global problem and essentially requires solving another MIP.



(a) The **Reduced Tree** representing environment 5.1e. The path through the graph representing the sequence of nodes to reach the end point (root) from the start point is highlighted.

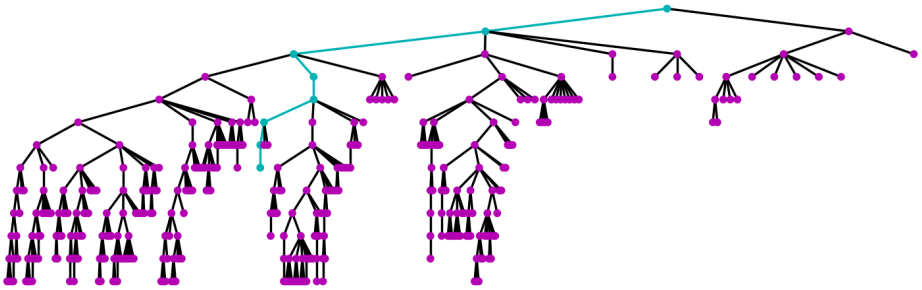


(b) The sequence of nodes through the graph laid out on the environment, represented as regions, with the start point in green and end point in red.

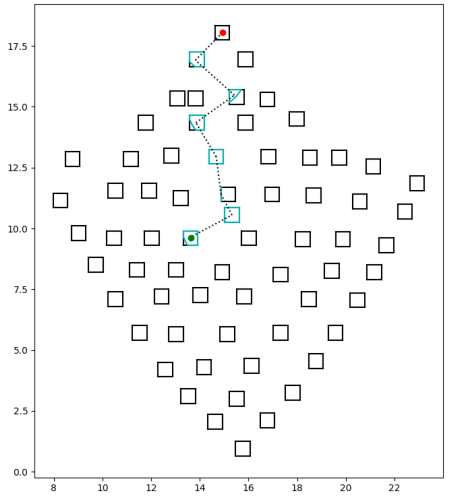


(c) The contact points generated by the graph-assisted MIP.

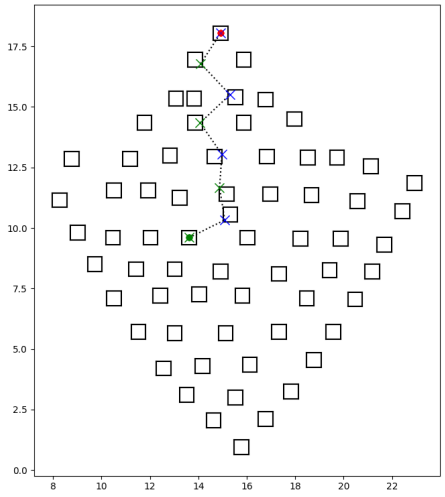
Figure 4.7: An illustration of the graph-assisted MIP process. From the start to the end point of the environment. ($r = 1.8, n = 16, \Delta = 0.1$, first contact = right)



(a) Another path through the graph seen in Figure 4.7a, this time representing the sequence of nodes to reach the end point (root) from the new start point is highlighted.

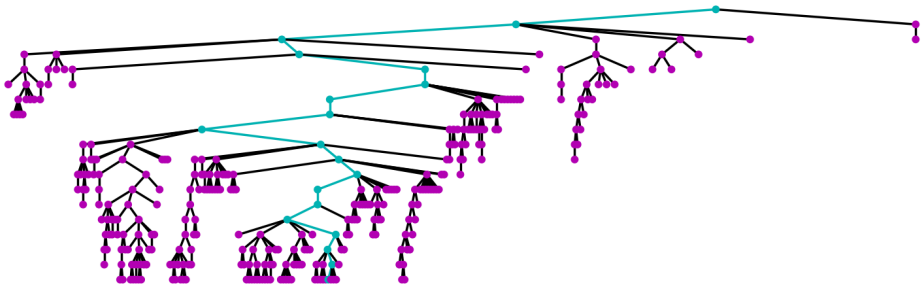


(b) The sequence of nodes through the graph from the new start point laid out on the environment, represented as regions, with the new start point in green and end point in red.

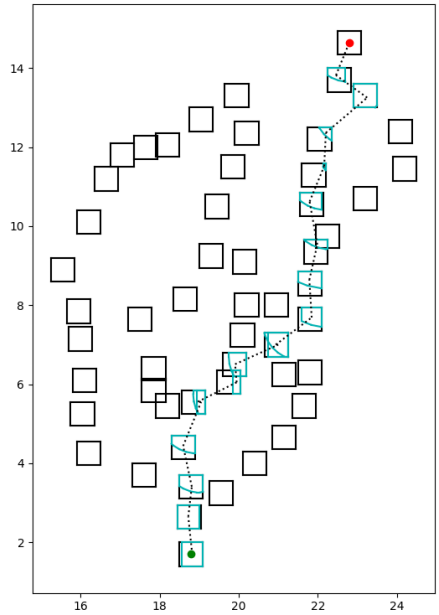


(c) The re-planned contact points generated by the graph-assisted MIP.

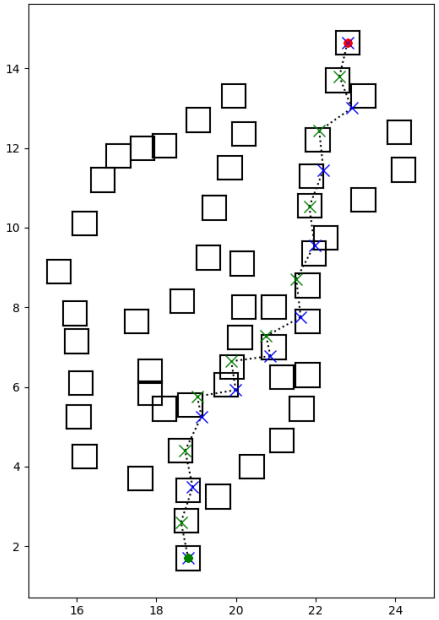
Figure 4.8: An illustration of the re-planning process of the graph-assisted MIP. A new start point is given that is not part of the original plan, but is part of the graph. Recall that this process can be done much faster than simply re-running the MIP. ($r = 1.8, n = 16, \Delta = 0.1$, first contact = right)



(a) A path through the **Reduced Tree** representing environment 5.1a from the start point to the end point.

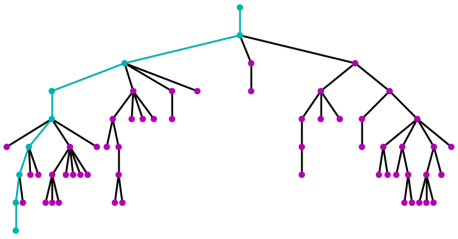


(b) The sequence of nodes through the graph from the new start point laid out on the environment, represented as regions, with the new start point in green and end point in red.

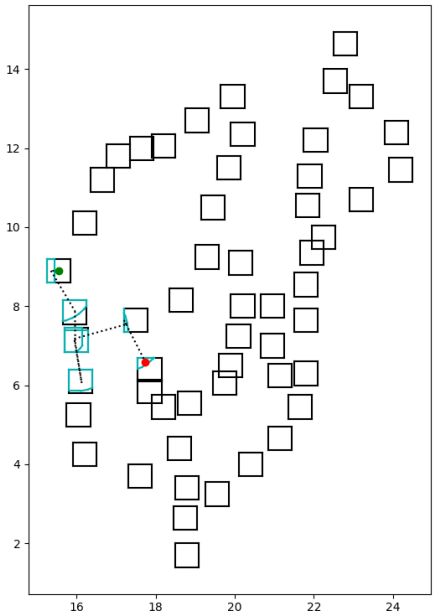


(c) The contact points generated by the graph-assisted MIP.

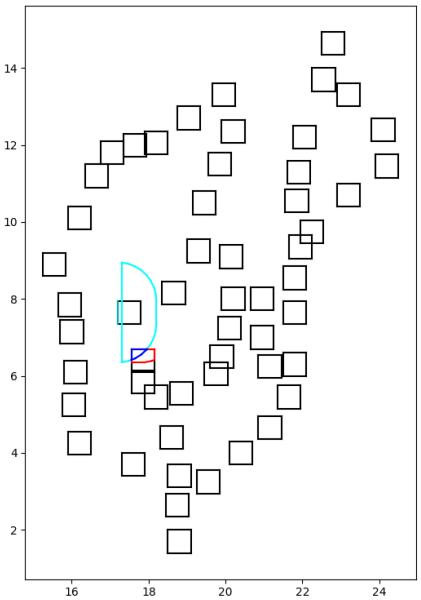
Figure 4.9: An illustration of the graph-assisted MIP process. From the start to the end point of the environment. ($r = 1.8, n = 16, \Delta = 0.1$, first contact = right). We introduce this figure to provide context for Figure 4.10.



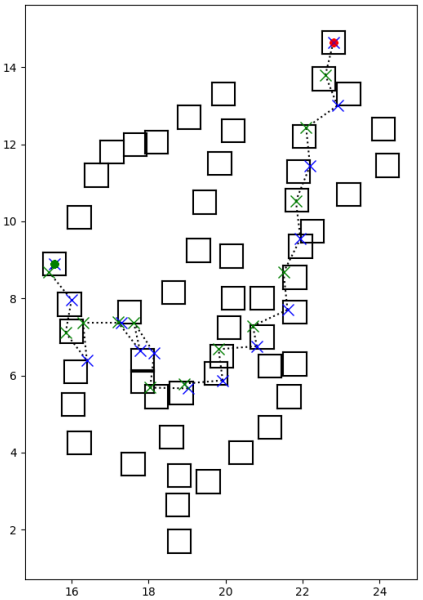
(a) The new graph constructed in order to carry out Procedure ii outlined in Section 4.3.1, with the path from the new start position (root) to the node intersecting the old graph highlighted.



(b) The new graph's path laid out as regions on the environment.



(c) An illustration of how exactly the new graph intersects the old graph. The red region is a node of the old graph.



(d) The contact points generated by the graph-assisted MIP after following Procedure ii in Section 4.3.1.

Figure 4.10: An example of the ability of the graph approach to "re-attach" itself to the graph in the event of an unexpected position outside of the graph. Note that we do not expect this to be a likely scenario as we reason in Section 4.3.1.

4.3.2 In the Event of Changing Objective

We assume that we have perfect knowledge of the node(s) containing the end point. If our end point changes, and is now represented by some other node in the graph part-way through locomotion, we are faced with the more general problem of graph traversal. Since the new goal is not to traverse to the root, we cannot rely on up-traversal to generate a path through the graph. Note that we have changed our end point as well as start point.

As we have full knowledge of the graph and knowledge of the goal node, we can perform a semi-informed search, in that we only need to look for paths to nodes with the same depth as the goal. If the new end point is contained within multiple nodes of varying depth, we can search each depth individually. The order of nodes we search can be heuristically set, such as by order of depth nearest the depth of our point, or by an A*-like approach [15], in which we take the heuristic to be the real distance between the centre point of the node and the new start point. Further testing would be required to show the efficacy of each of these heuristics.

It is very important to note the major limitation of downwards graph traversal in this context, that having a contact point inside a node is not sufficient to guarantee the reachability of its children. We must be able to guarantee that our reachable state fully includes the node's region we wish to traverse downwards from. This is not a problem with traversal to a parent node, as we can reason that there must be some point within the parent node that leads to the current node, and apply this logic to each parent node up to the root. In order to guarantee that the robot's reachable state entirely includes a node – and therefore can reach all of its children – we must make multiple steps within that region to "expand" our state. This limitation is the reason that this approach is not fully implemented.

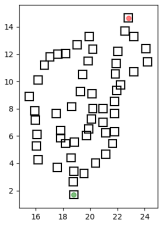
4.3.3 In the Event of Changing Environment

In a **Full Tree** or a **Depth-aware Graph**, it is likely that any given point in the environment is included in multiple nodes of varying depths. In reality, this can be thought of as there existing multiple possible paths to and from this point. This idea of alternate paths from a single point being encoded in the same graph gives rise to the possibility of re-planning in the event of changing environment *through up-traversal*. Recall that we assume the robot has perfect knowledge of which node(s) it is occupying.

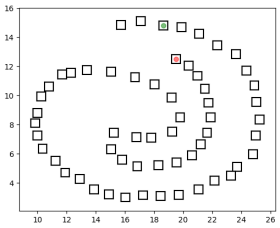
For instance, if part of an environment region is made invalid through introduction of an obstacle or otherwise, we purge any nodes within that region, thereby disconnecting all of their offspring from the root. Then, we take the lowest-depth, non-affected node that the robot is occupying as our current node. From there, we simply use up-traversal to find the path to the end-point. However, this procedure requires that such a node exists. If it does not, we must use a modified version of Procedure ii in Section 4.3.1 after purging the affected nodes. The modification is that, when constructing a new graph, we do not add any nodes which are created by the intersection of the walkable space with the now-invalid environment region. As it stands, this approach is theoretical, and requires further implementation and testing in order to confirm its efficacy.

Chapter 5

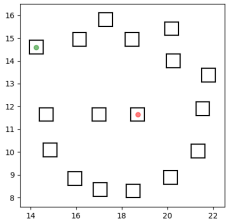
Experiments



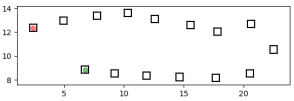
(a) An environment with a scattered array of walkable regions.



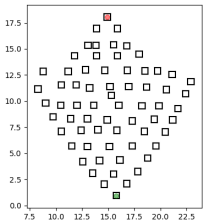
(b) An environment of a large number of regions arranged in a spiral fashion.



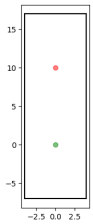
(c) An environment of a small number of regions arranged in a small spiral pattern.



(d) An environment with small walkable regions arranged in sideways U shape.



(e) An environment arranged with many walkable regions such that there are many paths that the robot can take to reach the end point.



(f) An environment with a single region, to demonstrate intra-region capabilities.

Figure 5.1: Six environments of varying complexity, representing a range of environments the robot may encounter. The start point is marked with a green point, and the end point is marked with a red point. The walkable regions are marked with black outlines

We hypothesise that providing the MIP with an optimal pre-computed set of binary variables significantly increases its performance. Our goal is to test the benefits of us-

ing the graph construction approach and determine when it is and is not appropriate to use compared to simply running the MIP from a new starting point. We will refer to the latter approach as the ”Pure MIP” approach, in contrast to the ”Graph-assisted MIP” approach. In terms of performance when solving the initial MIP, the time required to construct the graph is significantly longer than that needed to solve the pure MIP using the Gurobi solver. However, this outcome is expected for a preliminary Python prototype. This prototype serves as an initial proof of concept and should not be directly compared with a fully optimised solver like Gurobi. The prototype has not undergone thorough optimisation and refinement. For details on suggested optimisations for graph construction, see Section 7.1.1.

Figure 5.1 introduces a non-trivial handful of environments over which we aim to generate contact plans.

5.1 Replanning in the Event of Unexpected Position.

We are assumed to have constructed the graph already. This section aims to demonstrate the effectiveness of the graph with respect to quickly re-planning a route to the end point from various points in the environment using the same graph. We assume that the robot has perfect knowledge of which node(s) it is currently occupying.

We trivially expect the graph-assisted MIP to be solved faster than the pure MIP, as it is a more constrained problem. Our interest lies in how much faster this approach allows us to compute these paths. To generate the unexpected contact points, we set a new start point as the centre point of a random environment region, and randomise the foot which the contact point is associated with. As for the way we construct the graph, we choose the **Reduced Tree** method introduced in Section 4.2.2. This choice allows us to make few compromises on optimality of the new paths generated (Table 4.1), while allowing us a reasonable speed of graph construction.

We do not account for the time taken to perform up-traversal itself as, even in this implementation, it takes time on the order of 10 µs to find a path of 46 steps. If performance were to improve such that 10 µs is a non-negligible time, we would account for it.

5.1.1 Unexpected Position is Contained Within a Node of the Graph

We simulate a series of scenarios in which an unexpected contact point is contained within a node of the graph, necessitating the generation of another contact plan to reach the end point of the environment. In this scenario, the foot producing the unexpected contact point is the foot associated with the new node, so no additional computation is required and traversal is trivial from the new position.

We compare the time to solve the graph-assisted MIP vs the time to solve the full MIP from the new starting point. Our metric will be the time taken to run the model’s `optimize()` method as measured with Python’s `perf_counter`. There exists a debug output by the solver itself which provides the time taken to solve, but it is not granular

enough for our needs, as it outputs `0.00 seconds` or `0.01 seconds` in the majority of cases.

5.1.2 Unexpected position is Not Within a Node

If the robot ends up in a position that is outside the graph, as covered in Section 4.3.1, recall that we start a new graph and iterate until one of our nodes intersects with a node in the original graph. From there, we use up-traversal to find a path from the root of the new graph to that node, then from that node to the root of the original graph, also using up-traversal. This path through both graphs will correspond to a path from the new position to the end point. Additionally, as reasoned in Section 4.3.1, it is likely that our unexpected position is close to a node in the graph, meaning the new graph we construct may not need many steps before it reaches the original graph, and will therefore be theoretically fast to compute. However, since we randomise our position, this does not apply in our results. We use the **Reduced Tree** technique to construct this new graph.

We do not expect the construction of the new graph to be faster than the Pure MIP approach, due to reasons listed at the beginning of this chapter. However, if only a small number of steps is required, we expect the graph construction speed to at least be on par with the Pure MIP approach.

We compare the sum of the the time to construct the new graph and the time to solve the graph-assisted MIP to the time it takes to solve the full MIP from the new starting point. Our metric will be the time taken to run the model's `optimize()` method as measured with Python's `perf_counter`, as well as the time taken for the implemented `create_new_graph_in_search_of_other_graph()` function to return the nodes of the new graph, also using `perf_counter`.

Of our six test environments (Figure 5.1), only two environments contained feasible positions not explored by the **Reduced Tree**, being the environments featured in figures 5.1a and 5.1b. Hence, we can only get results for these two environments. To improve this experimental procedure, we would include more test environments with start and end points that are not on the extremities of the environment in order to ensure that there are some regions that are not explored.

Chapter 6

Results

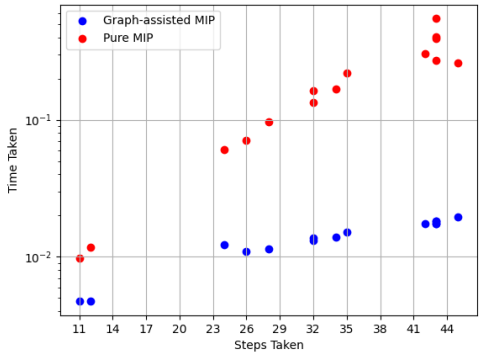
We present the results of our experiments comparing the performance of the Graph-assisted MIP approach and the Pure MIP approach in terms of time required to compute contact plans in various environments. As previously mentioned, our primary interest lies in how much faster the Graph-assisted MIP approach allows us to compute the paths compared to the Pure MIP approach after computation of the graph itself. We discuss the results based on the two scenarios outlined in the experiments chapter: 1) Unexpected position in a node of the graph, and 2) Unexpected position not in graph.

6.1 Replanning from Inside Graph

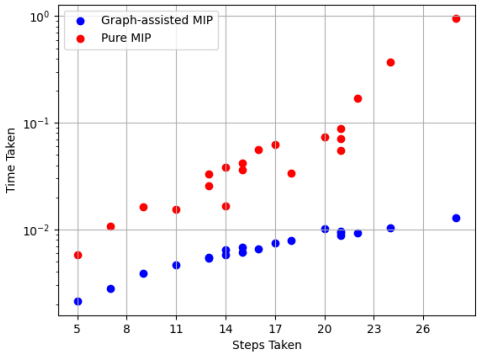
In this scenario, we found that the Graph-assisted MIP approach consistently outperformed the Pure MIP approach across all test environments. The time required to compute contact plans using the Graph-assisted MIP approach was significantly lower than that of the Pure MIP approach, with the difference between times increasing exponentially as the number of steps increased, as shown in 6.1.

6.2 Replanning from Out of Graph

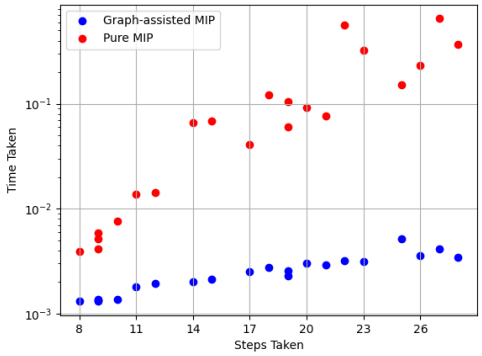
For the cases where the unexpected position was not contained within the graph constructed to find a path between the original start and end points, we observed that the performance difference between the Graph-assisted MIP and the Pure MIP approach was heavily dependent on environment. In environment 5.1a, the pure MIP consistently solved the MIP faster than the new graph is constructed. However, in environment 5.1b, the time taken to construct the graph and generate a contact plan with the Graph-assisted MIP was actually lower than that of the Pure MIP as shown in Figure 6.2. We hypothesise that this is due to positions of the only regions not covered by the graph being very close to the original graph, meaning that the new graph constructed would not need to be computed to a high depth before intersecting with the original graph.



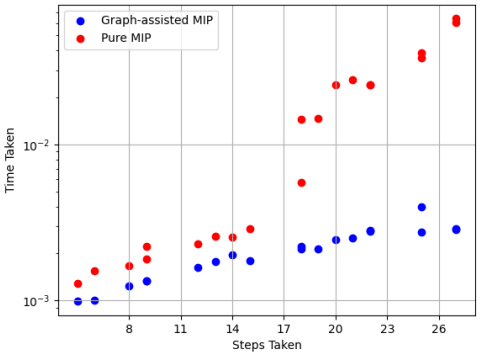
(a) Environment 5.1a, 26.5 seconds



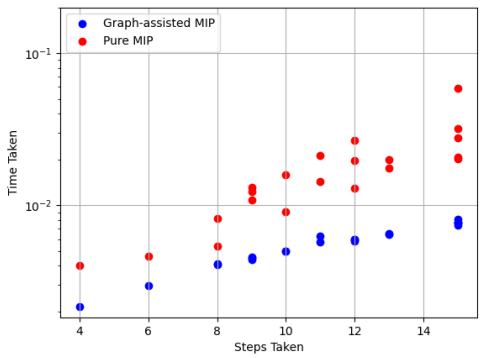
(b) Environment 5.1b, 28.5 seconds



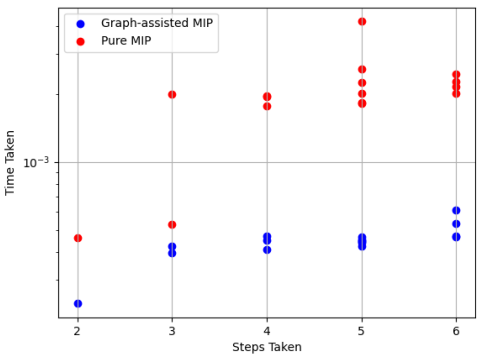
(c) Environment 5.1c, 2.2 seconds



(d) Environment 5.1d, 2.1 seconds

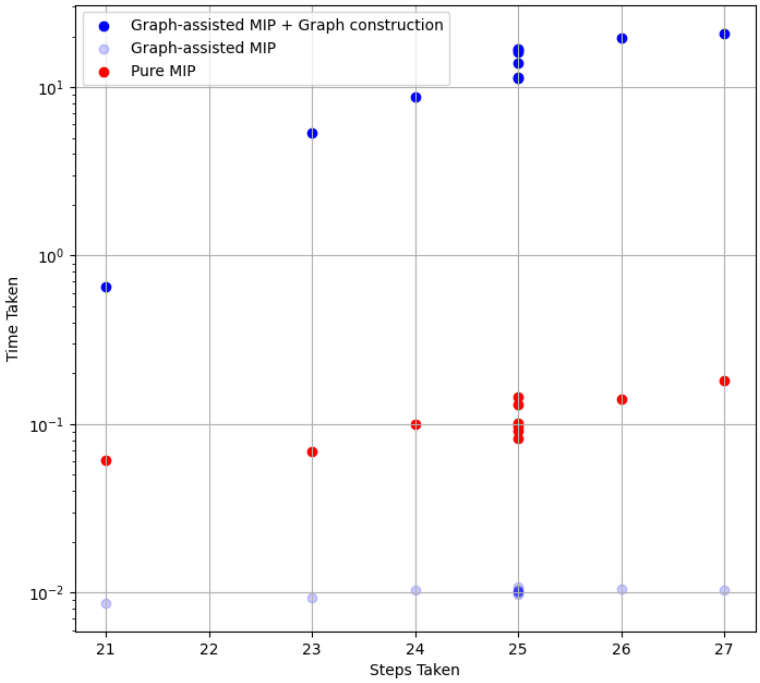


(e) Environment 5.1e, 22.1 seconds

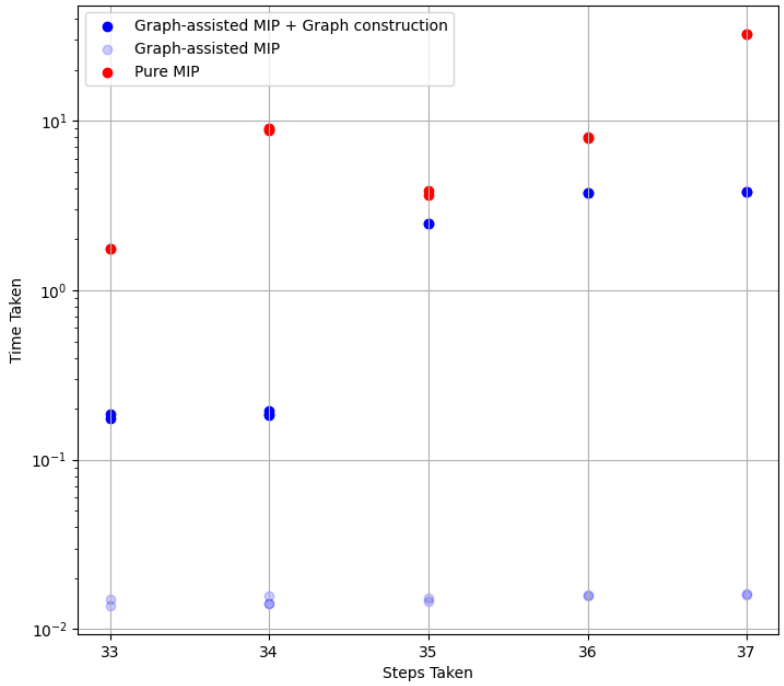


(f) Environment 5.1f, 0.04 seconds

Figure 6.1: Times taken to plan a route from a point included in the graph by graph-assisted MIP vs Pure MIP. The time taken to construct the graph is in the subcaption of each subfigure. Note the Log scale on the y-axis.



(a) Environment 5.1a.



(b) Environment 5.1b.

Figure 6.2: Times taken to create a contact plan to the end point from a point not included in the graph by performing Procedure ii from Section 4.3.1 vs Pure MIP. Note the Log scale on the y-axis.

Chapter 7

Discussion

7.1 Further Work

7.1.1 Performance of Graph Construction

For this paper, the graph construction algorithm is implemented in Python and many avenues for optimisation are not yet explored.

For instance, we can draw many similarities between our graph construction approach and the canonical graph traversal approach of Breadth First Search (BFS), in the sense that every node of depth n is computed before computing any nodes at depth $n+1$. Parallel BFS algorithms have been shown to be successful in improving performance over Serial BFS [2]. Through this, we conjecture that this is an excellent opportunity to implement multi-threading, and that taking a multi-threaded approach to constructing the graph would improve performance by allowing computation of nodes to be done in parallel, since each node is computed independently of all other nodes except its ancestors. Bear in mind that, of the proposed construction approaches in 4.2, this independence is only true in the case of the **Full Tree** construction approach. The faster approaches involve pruning of the tree as it is constructed, which involves comparison between nodes which are computed at the same step. Hence, multi-threading construction of the **Reduced Tree** and **Depth-aware Graph** requires some extra caution. We nevertheless conjecture that it would provide a performance boost over serial graph construction.

7.1.2 Extension of the formulation into 3 dimensions

As it stands, there are a few logical next steps to increase the scope of this work, such as further work on pruning the tree, performance improvements, or more robust graph-traversal algorithms which can handle down-traversal of the tree by default (see Section 4.3.2). One such next step is to extend our approach to a 3-dimensional workspace. In fact, the formulation proposed in the state-of-the-art works in three dimensions [31]. This would increase the robustness of our model, as it would be able to handle flat terrain at varying heights in a single environment, as opposed to strictly flat terrain

constrained to a single 2-dimensional plane, which is the current scope of our model.

7.2 Conclusion

We provide a brief critical review of the work presented. Firstly, the implementation provided does not account for rotation of the foot, nor does it immediately extend to 3-dimensional space. Furthermore, we speculate that the addition of these features would increase the branching factor of the graph construction approach such that it becomes infeasible to compute to any useful depth. Hence, we must rely on further work to extend our ideas to a point at which they are applicable in the real world. The results of the first experiment clearly show that the graph approach succeeds in its goal. However, one could argue against its effectiveness as a whole, as the time taken for initial graph construction is much greater than the Pure MIP approach. We argue that the graph construction can be made significantly faster through optimisations in the following section. However, this again relies on further work and investigation.

To conclude, the main achievements of this work are as follows:

- A Mixed-Integer formulation was constructed to provide a contact plan for a bipedal robot.
- A new contribution was proposed in the form of a specific environment abstraction which relies on the assumptions laid out for the Mixed-Integer formulation.
- An original implementation of both of the above was written in Python for this dissertation and experimented upon.
- A performance based argument, backed up by data, was made for the benefits of the proposed new approach.

Bibliography

[1] S.P. Anbuudayasankar, K. Ganesh, and K. Mohandas. "Mixed-Integer Linear Programming for Vehicle Routing Problem with Simultaneous Delivery and PickUp with Maximum Route-Length". In: *The International Journal of Applied Management and Technology* 6.1 (). URL: https://scholarworks.waldenu.edu/cgi/viewcontent.cgi?article=1020&context=ijamt.

[2] David A. Bader and Kamesh Madduri. "Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2". In: *2006 International Conference on Parallel Processing (ICPP'06)*. 2006, pp. 523–530. DOI: 10.1109/ICPP.2006.34.

[3] Léo Baudouin et al. "Real-time Replanning Using 3D Environment for Humanoid Robot". In: *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS 2011)*. Oct. 2011, pp. 584–589. URL: https://hal.science/hal-00601300/document.

[4] Dimitri P. Bertsekas. *Convex Optimization Theory*. Athena Scientific, 2009. URL: http://web.mit.edu/dimitrib/www/Convex_Theory_Entire_Book.pdf.

[5] Aude Billard and Danica Kragic. "Trends and challenges in robot manipulation". In: *Science* 364 (6446 2019). DOI: 10.1126/science.aat8414. URL: https://www.science.org/doi/full/10.1126/science.aat8414.

[6] Dionysis D. Bochtis, Claus G.C. Sørensen, and Patrizia Busato. "Advances in agricultural machinery management: A review". In: *Biosystems Engineering* 126 (Oct. 2014), pp. 69–81. DOI: https://doi.org/10.1016/j.biosystemseng.2014.07.012.

[7] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK; New York: Cambridge University Press, 2004. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[8] Justin Carpentier and Pierre-Brice Wieber. "Recent Progress in Legged Robots Locomotion Control". In: *Current Robotics Reports* 2 (2021), pp. 231–238. DOI: 10.1007/s43154-021-00059-0.

[9] Robin Deits and Russ Tedrake. "Computing large convex regions of obstacle-free space through semidefinite programming". In: *Workshop on the Algorithmic Foundations of Robotics*. Istanbul, Turkey, 2014. URL: http://groups.csail.mit.edu/robotics-center/public_papers/Deits14.pdf.

[10] Farbod Farshidian et al. "Real-time motion planning of legged robots: A model predictive control approach". In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. 2017, pp. 577–584. DOI: 10.1109/HUMANOIDS.2017.8246930.

[11] Robert J. Griffin et al. “Footstep Planning for Autonomous Walking Over Rough Terrain”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. 2019, pp. 9–16. DOI: 10.1109/Humanoids43949.2019.9035046.

[12] Igor Griva, Stephen G. Nash, and Ariela Sofer. *Linear and Nonlinear Optimization (2. ed.)*. SIAM, 2008, pp. I–XXII, 1–742. ISBN: 978-0-89871-661-0.

[13] *Gurobi Optimization*. 2023. URL: https://www.gurobi.com.

[14] Mathew Halm and Michael Posa. “A Quasi-static Model and Simulation Approach for Pushing, Grasping, and Jamming”. In: *GRASP Laboratory* (2018). URL: https://dair.seas.upenn.edu/assets/pdf/Halm2018.pdf.

[15] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[16] Armin Hornung et al. “Anytime search-based footstep planning with suboptimality bounds”. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. 2012, pp. 674–679. DOI: 10.1109/HUMANOIDS.2012.6651592.

[17] *How does presolve work?* Gurobi Support. 2023. URL: https://support.gurobi.com/hc/en-us/articles/360024738352-How-does-presolve-work-.

[18] Qiang Huang et al. “Planning walking patterns for a biped robot”. In: *IEEE Transactions on Robotics and Automation* 17.3 (2001), pp. 280–289. DOI: 10.1109/70.938385.

[19] IBM. *MIQCP: Mixed Integer Programs with Quadratic Terms in Constraints*. Accessed: 2023-04-02. IBM. 2021. URL: https://www.ibm.com/docs/en/icos/22.1.1?topic=smippqt-miqcp-mixed-integer-programs-quadratic-terms-in-constraints.

[20] International Federation of Robotics. *The Impact of Robots on Productivity, Employment and Jobs*. Positioning Paper. International Federation of Robotics, Apr. 2017.

[21] *Introduction to Optimization Models*. 2011. URL: https://web.stanford.edu/~ashishg/msande111/notes/chapter3.pdf.

[22] Fotios Katsilieris. “Search and secure using mobile robots”. In: (Mar. 2023).

[23] Steven M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. Technical Report TR 98-11. Computer Science Department, Iowa State University, Oct. 1998.

[24] Peter Leven and Seth Hutchinson. “A Framework for Real-time Path Planning in Changing Environments”. In: *The International Journal of Robotics Research* 21.12 (2002), pp. 999–1030. DOI: 10.1177/027836490202101 2001. eprint: https://doi.org/10.1177/027836490202101 2001. URL: https://doi.org/10.1177/027836490202101 2001.

[25] Shengchen Liu. *Approximate Cell Decomposition*. Accessed: 2023-04-12. 2023. URL: https://shengchen-liu.github.io/robotics_gitbook/path_planning/approximate_cell_decomposition.html.

[26] *Matplotlib: Visualization with Python*. 2023. URL: https://matplotlib.org.

[27] P. Michel et al. “Vision-guided humanoid footstep planning for dynamic environments”. In: vol. 2005. Jan. 2006, pp. 13–18. ISBN: 0-7803-9320-1. DOI: 10.1109/ICHR.2005.1573538.

[28] *NetworkX*. 2023. URL: https://networkx.org.

[29] Koichi Nishiwaki et al. “The experimental humanoid robot H7: a research platform for autonomous behaviour”. In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 365 (Feb. 2007), pp. 79–107. DOI: 10.1098/rsta.2006.1921.

[30] Eric Pairet et al. “Online mapping and motion planning under uncertainty for safe navigation in unknown environments”. In: *IEEE Transactions on Automation Science and Engineering* 19.4 (2022), pp. 3356–3378. DOI: 10.1109/tase.2021.3118737.

[31] Russ Tedrake Robin Deits. “Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization”. In: (Nov. 2014).

[32] M. G. Tamizi, M. Yaghoubi, and H. Najjaran. “A review of recent trend in motion planning of industrial robots”. In: *International Journal of Intelligent Robotics and Applications* (2023). DOI: 10.1007/s41315-023-00274-2.

[33] Steve Tonneau et al. “SL1M: Sparse L1-norm Minimization for contact planning on uneven terrain”. In: *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[34] Various. *Linear and Quadratic Programming: Half Spaces*. Accessed: 2023-04-09. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. 2021. URL: https://inst.eecs.berkeley.edu/~ee127/sp21/livebook/l_lqp_half_spaces.html.

[35] Various. *Mixed Integer Programming*. Accessed: 2023-04-09. n.d. URL: https://www.sciencedirect.com/topics/computer-science/mixed-integer-programming.

[36] Martin Wermelinger et al. “Navigation planning for legged robots in challenging terrain”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1184–1189. DOI: 10.1109/IROS.2016.7759199.

[37] Takanobu Yamamoto and Tomomichi Sugihara. “Responsive navigation of a biped robot that takes into account terrain, foot-reachability and capturability”. In: *Advanced Robotics* 35 (Mar. 2021), pp. 1–15. DOI: 10.1080/01691864.2021.1896382.