HYBRID PLANNING FOR FREE CLIMBING ROBOTS: COMBINING TASK AND MOTION PLANNING WITH DYNAMICS AND CONTROL

by
EMRE CEMAL GÖNEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2021

HYBRID PLANNING FOR FREE CLIMBING ROBOTS: COMBINING TASK AND MOTION PLANNING WITH DYNAMICS AND CONTROL

Approved by:

Date of Approval: Jul 13, 2021

ABSTRACT

HYBRID PLANNING FOR FREE CLIMBING ROBOTS:
COMBINING TASK AND MOTION PLANNING
WITH DYNAMICS AND CONTROL

EMRE CEMAL GÖNEN

MECHATRONICS ENGINEERING M.Sc. THESIS, JULY 2021

Thesis Supervisor: Prof. Dr. Volkan Patoğlu
Thesis Co-Supervisor: Prof. Dr. Esra Erdem

Keywords: Hybrid planning, answer set programming, motion planning, multi-contact locomotion of legged robotics, inverse dynamics control, free climbing robots

Robots with articulated limbs that can free-climb vertical surfaces have the potential to be instrumental in a wide range of applications, ranging from search-and-rescue to surveillance, inspection/maintenance to planetary exploration. Free climbing is highly challenging as it requires the robot to make progress using only friction at the contact points, without using any special equipment.

To free-climb vertical terrain, the robot must go through a continuous sequence of configurations satisfying certain constraints that ensure that the robot is in equilibrium, collision-free, and can be controlled within the actuator torque limits. A feasible free-climb plan requires i) deciding on a proper sequence of holds to reach, ii) finding collision-free trajectories for the relevant arms of the robot reach these holds, while utilizing the internal degrees of freedom of the robot to maintain friction contacts and balance of the robot, and iii) ensuring that these trajectories can be executed within the actuator torque limits. Therefore, geometric reasoning and motion planning alone are not sufficient to solve these problems, as the planning of reach actions need to be integrated with the motion planning and the feasibility of plan executions in terms of maintaining friction contacts, balance and actuation capabilities needs to be verified.

We propose a hybrid planning approach for free climbing robots that combines high-level representation and reasoning with low-level geometric reasoning, motion planning, balance, and actuator feasibility checks. The hybrid planning approach features bilateral interaction between high-level reasoning and feasibility checks. The high-level reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic/dynamic/controls solution for that task-plan, then the feasibility checks guide the high-level reasoner by modifying the planning problem with new constraints. We present a validation of our approach through a comprehensive set of benchmark instances and a systematic evaluation its performance in terms of scalability, solution quality, and success rate.

ÖZET

SERBEST TIRMANAN ROBOTLAR İÇİN HİBRİT PLANLAMA:
GÖREV VE HAREKET PLANLAMANIN
DİNAMİK VE KONTROL İLE BİRLEŞTİRİLMESİ

EMRE CEMAL GÖNEN

MEKATRONİK MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, TEMMUZ 2021

Tez Danışmanı: Prof. Dr. Volkan Patoğlu
Tez Eş Danışmanı: Prof. Dr. Esra Erdem

Anahtar Kelimeler: Hibrit planlama, çözüm kümesi programlama, hareket planlama, bacaklı robotların çok temaslı hareketi, ters dinamik kontrolü, serbest tırmanan robotlar

Dikey yüzeylere serbest tırmanabilen eklemli uzuvlara sahip robotlar, arama-kurtarmadan gözetlemeye, denetim/bakımdan gezegen keşfine kadar çok çeşitli uygulamalarda faydalı olma potansiyeline sahiptir. Serbest tırmanma, robotun herhangi bir özel ekipman kullanmadan sadece temas noktalarındaki sürtünmeyi kullanarak ilerleme kaydetmesini gerektirdiğinden oldukça zordur.

Dikey arazide serbest tırmanmak için, robotun dengede olmasını, çarpışmasız olmasını ve eyleyici tork limitleri dahilinde kontrol edilebilmesini sağlayan belirli kısıtlamaları karşılayan sürekli bir konfigürasyon dizisinden geçmesi gerekir. Uygulanabilir bir serbest tırmanma planı şu aşamaları gerektirir: i) ulaşmak için uygun bir tutacak sırasına karar vermek, ii) robotun ilgili kollarının bu tutacaklara ulaşması için çarpışmasız yörüngeler bulmak ve aynı zamanda robotun sürtünme temaslarını ve dengesini korumak için dahili serbestlik derecelerini kullanmak ve iii) bu yörüngelerin eyleyici tork limitleri dahilinde gerçekleştirilebilmesini sağlamak. Bu nedenle, geometrik muhakeme ve hareket planlama tek başına bu problemleri çözmek için yeterli değildir; erişim eylemlerinin planlanması hareket planlama problemi ile bütünleştirilmelidir ve sürtünme temaslarını sürdürme açısından planın fizibilitesi doğrulanmalıdır.

Bu çalışmada serbest tırmanan robotlar için yüksek seviyeli bilgi gösterimi ve otomatik akıl yürütme ile düşük seviyeli geometrik muhakeme, hareket planlama, denge ve eyleyici fizibilite kontrollerini birleştiren melez bir planlama yaklaşımı öneriyoruz. Melez planlama yaklaşımı, üst düzey akıl yürütme ve fizibilite kontrolleri arasındaki karşılıklı etkileşimi içerir. Üst düzey akıl yürütücü, optimal bir görev planı bularak hareket planlayıcıya rehberlik eder. Hareket planlayıcı, o görev planı için uygun bir kinematik/dinamik/kontrol çözümü yoksa planlama problemini yeni kısıtlarla değiştirerek üst düzey akıl yürütücüye rehberlik eder. Önerilen melez planlama yaklaşımının ölçeklenebilirliği, çözüm kalitesi ve başarımı bir dizi örnek senaryo vasıtasıyla sistematik olarak incelenmiştir.

ACKNOWLEDGEMENTS

First, I would like to thank my mother and my father, Huriye and Yaşar, and my sisters Elif and Esra. I am indebted to you as you have always been there when I need help.

I owe my deepest gratitude to Prof. Dr. Volkan Patoğlu and Prof. Dr. Esra Erdem for their guidance and attention. They suggested an exciting and challenging thesis topic. They showed great confidence in me while always making sure I was moving in the right direction throughout this research.

I would like to acknowledge the Scientific Technological Research Council of Turkey (TÜBİTAK) for partially supporting this work under grant 118E931.

It is a pleasure to thank professors and researchers at Sabancı University. It has been an honor being alumni of such an excellent institute. I appreciate all opportunities they provided for me.

I am grateful to my lab mates that are Çağatay, Müge, Vatan, Emre, Aysu, and Çağrı. I also owe many gracias to Ali, Uğur, and Cansu, who shared stressful times with me. My old friends, Akif, İshak, Furkan, Emre, and Kaan, have always been my connection to the outer world during the COVID-19 pandemic. Of course, I cannot forget Kadir, Ali, Çağrı, Görkem, Okan, Önder and エブルさま (Ebru). They shared their experiences and let me be informed about other research activities.

And last but not least, I would like to thank *mio caro* Zehra for always being present in my life. Her endless support and understanding have been the moral support throughout this research. 사랑해야...

To the future or to the past, to a time when thought is free, when men are different from one another and do not live alone; to a time when truth exists and what is done cannot be undone...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Robots with articulated limbs have multiple arms connected to their bodies. Each arm has multiple degrees of freedom and does not have any mechanical gripper, attachment mechanism, or adhesive material placed at its ends.

Robots with articulated limbs that can free-climb vertical surfaces have the potential to be instrumental in a wide range of applications, ranging from search-and-rescue to surveillance, inspection/maintenance to planetary exploration (Luk, Collie & Billingsley, 1991; Nagakubo & Hirose, 1994; Xiao, Dulimarta, Yu, Xi & Tummalal, 2000; Xiao, Sadegh, Elliot, Calle, Persad & Chiu, 2005).

On the other hand, free-climbing is highly challenging for robots, as it requires a robot to make progress using only friction at the contact points without using any special equipment. Thus, finding a feasible free-climb plan is challenging.

In particular, a feasible free-climb plan for a robot with articulated limbs requires

(i) deciding on a proper sequence of holds to reach,

(ii) finding collision-free trajectories for the relevant arms of the robot to reach these holds, while utilizing the internal degrees of freedom of the robot to maintain friction contacts and balance of the robot, and

(iii) ensuring that these trajectories can be executed within the actuator torque limits.

Therefore, computing a free-climb plan requires task planning, motion planning, geometric reasoning, and feasibility checks. Furthermore, the feasibility of plan executions in maintaining friction contacts, balance, and actuation capabilities needs

to be verified. We propose a novel hybrid planning method for free climbing robots that combines high-level representation and reasoning (to address (i) above) with low-level geometric reasoning, motion planning, balance, and actuator feasibility checks (to address (ii) and (iii) above).

The hybrid planning approach features bilateral interaction between high-level reasoning and feasibility checks. The high-level reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic/dynamic/controls solution for that task-plan, then the feasibility checks guide the high-level reasoner by modifying the planning problem with new constraints.

In particular, the hybrid planning method is novel and addresses significant challenges as follows:

- Classical task planning computes a discrete plan (i.e., a sequence of actions) of a given length to reach a goal state from an initial state. The task planning method introduced in this thesis considers the robot's moves as actions, positions and poses as states. It aims to find a **shortest** plan that the robot can follow to reach the goal position from its initial position. Furthermore, it considers a set of **waypoints** for the robot to visit on its way to the goal position, as well as some **obstacles** (e.g., some regions) to avoid. In addition to minimizing the plan length (i.e., makespan), the task planning method maximizes a given **cost function** that characterizes the robustness of the plan.

  We solve such a complex task planning problem by reducing it to a graph problem and use Answer Set Programming (ASP) (Brewka, Eiter & Truszczynski, 2016; Gelfond & Lifschitz, 1991; Lifschitz, 2002a; Marek & Truszczyński, 1999; Niemelä, 1999)—an AI paradigm based on declarative problem representation and automated reasoning—and the ASP solver Clingo (Gebser, Kaminski, Kaufmann & Schaub, 2019) with multi-shot computation to solve it.

- Feasibility checks are an essential component of our hybrid planning method for free-climbing multi-limbed robots. Our novel graph-based method for task planning utilizes stability and reachability checks to identify the feasible stances and the feasible transitions between two poses.

- Motion planning computes a continuous collision-free trajectory from an initial configuration of a robot to its final configuration. Our hybrid method considers a free-climbing robot as a planar parallel manipulator as if each leg is fixed to the contact points, takes into account collisions with obstacles and self-collisions, and utilizes motion planning (with geometric constraints for the closed-loop kinematic chain) to find a collision-free trajectory for each action of the discrete task plan. In this way, the robot decides how it should move its body and arms from the current stance to the next stance to execute an action.

  We use sampling-based motion planning (Kavraki & LaValle, 2016; Kavraki, Svestka, Latombe & Overmars, 1996b), and the motion planners available at the Open Motion Planning Library (OMPL) (Sucan, Moll & Kavraki, 2012).

- Our hybrid method also ensures that the continuous trajectories computed by the motion planner can be executed by robots under an inverse dynamics controller while obeying the actuator torque limits.

- If the motion planner cannot find a collision-free trajectory for some action in the task plan within the give time, or if the joint torque requirements for the actuators are too high for the feasibility of plan execution, then replanning is utilized by our hybrid planning method.

Another important novelty of our hybrid planning method is its flexibility to be extended to multiple free-climbing multi-limbed robots.

We analyze our hybrid method empirically by designing and generating a comprehensive set of benchmark instances for free-climbing multi-limbed robots and then systematically evaluating its performance in terms of scalability, solution quality, and success rate.

Thesis outline In the rest of the thesis, Chapter 2 discusses the related work in comparison with our studies. After providing preliminaries in Chapter 3, Chapter 4 presents our hybrid planning method for free-climbing robots. Chapter 5 illustrates this method with some interesting scenarios and presents a comprehensive empirical analysis of it. Chapter 6 concludes with discussions of our contributions and future work.

2. RELATED WORK

We consider related work in two groups: hybrid planning, where high-level reasoning is combined with low-level feasibility checks to ensure the feasibility of robot plan execution, and free-climbing robots and their planning.

Related work on hybrid planning.

Hybrid planning aims to combine classical task planning with motion planning to compute feasible plans for robots. None of the existing related work on hybrid planning aims for free-climbing robots.

The related work on hybrid planning have three main computational approaches: (i) developing/modifying search algorithms (Gravot, Cambon & Alami, 2003; Hauser & Latombe, 2009; Kaelbling & Lozano-Pérez, 2013; Kingston, Wells, Moll & Kavraki, 2020), (ii) utilizing formal methods (Dantam, Kingston, Chaudhuri & Kavraki, 2016; Plaku, 2012), and (iii) formally embedding motion planning as part of representations of actions (Caldiran, Haspalamutgil, Ok, Palaz, Erdem & Patoglu, 2009; Erdem, Haspalamutgil, Palaz, Patoglu & Uras, 2011; Erdem, Patoglu & Schüller, 2016; Gaschler, Petrick, Giuliani, Rickert & Knoll, 2013; Hertle, Dornhege, Keller & Nebel, 2012). Our approach is similar to the last one. We utilize, in particular, PRE (computation) method of Erdem et al. (2016) where feasibility checks are computed and then embedded in high-level representations.

Related work on learning-based approaches.

Several problems of robotics can be addressed using reinforcement learning to enable a robot to autonomously discover an optimal behavior over trial-and-error interactions with its environment (Kober, Bagnell & Peters, 2013). In reinforcement learn-

ing, the problem is designed such that a task provides feedback as a scalar objective function measuring the performance of the robot, which does not involve explicitly detailing the solution to the problem. This method is helpful to be used in a wide range of applications from aerial vehicles, robotic arms, autonomous vehicles, and humanoid robots (Bagnell & Schneider, 2001; Gullapalli, Franklin & Benbrahim, 1994; Mahadevan & Connell, 1992; Schaal, 1997).

Learning-based models can be used for generating policies for path planning, sometimes with waypoints and sometimes with obstacles (Blum, Jones & Yoshida, 2019; Lee, Shen, Yu, Singh & Ng, 2006). For instance, Lee et al. (2006) developed a hierarchical reinforcement learning approach using a two-level decomposition of task. First, a low-level controller generates the continuous motion to move each foot to specified positions using the policy search. During learning, a reward function that penalizes undesirable behaviors such as taking a long time to complete the foot movement, passing too close to the vertical surface of an obstacle, or failing to move the foot to the desired goal location is used. Second, a high-level controller selects the sequence of foot placement positions with a look ahead search using the cost estimate of the value function relative to the supervised data. Blum et al. (2019) considers a set of waypoints and picks the best one to step by with respect to the rewards function learnt by letting the robot's joints to explore states. Learning based methods can also be beneficial to determine gait patterns in dynamic legged locomotion can also be beneficial to determine gait patterns in dynamic legged locomotion (Kirchner, 1998; Manoonpong, Parlitz & Wörgötter, 2013). Improvement of the stability of locomotion on various terrains and obtaining a successful locomotion model for robots that are deployed in unpredictable terrains with geometries are some other applications on legged robots (Schilling, Konen & Korthals, 2020; Silva, Perico, Homem, Vilao, Tonidandel & Bianchi, 2015; Tsounis, Alge, Lee, Farshidian & Hutter, 2020).

On the other hand, climbing robot problems solved by learning-based models are usually defined as hill-climbing, step climbing, stair climbing, or climbing over obstacles (Schilling et al., 2020; Totani, Sato & Morita, 2019; Tsounis et al., 2020; Vincent & Sun, 2012; Wiley, Bratko & Sammut, 2017). The reward function is de-

fined as forward or vertical velocity of the robot with penalties of joint torques and impact forces. Therefore, the agent is forced move forward or climb upward while satisfying the feasibility. Among these problems, the closest climbing problem to the problem addressed in this thesis study is problem of a humanoid model climbing on a climbing wall (Garg, 2018). In this work, the authors aimed to train a humanoid simulation to climb a wall containing climbing holds as presented in Figure 2.1. The robotic agent was trained for 12 hours using Trust Region Policy Optimisation (TRPO) and for 42 hours using Deep Deterministic Policy Gradient (DDPG) (Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver & Wierstra, 2015; Schulman, Levine, Abbeel, Jordan & Moritz, 2015). Although both methods took 1 million time steps, and the researchers tried other configurations and modifications of the reward function, it was noted that these methods were unable to train the agent to climb. It might be possible to obtain successful results for a longer training time. However, considering the amount of time for training, our hybrid planning method is advantageous because we divide the problem into subcategories and use proper methods to solve them.

In general, our proposed work computes a plan using Task Planing on a feasibility graph, whereas the learning based approaches computes a policy with respect to a reward function. For that reason, our method engages the capabilities of high-level reasoning (i.e. task planning) as well as low-level reasoning (i.e. motion planning). In particular, with hybrid planning, our method can handle a given set of waypoints and instead of choosing which waypoint to visit on the way to the goal, it decides the order of all waypoints to visit the way to the goal. The latter challenge cannot be handled be handled by the learning based methods. Furthermore, our method allows us to minimize the total length/cost of the plan. This is not possible by action selector with lookahead search.

Related work on free-climbing robots.

For the last two decades, many robots have been designed and developed for climbing purposes. Most of them rely on their operational environment and climbing tools , and are usually equipped with grippers (Parness, Abcouwer, Fuller, Wiltsie, Nash & Kennedy, 2017), vacuums (Brockmann, Albrecht, Borrmann & Elseberg,

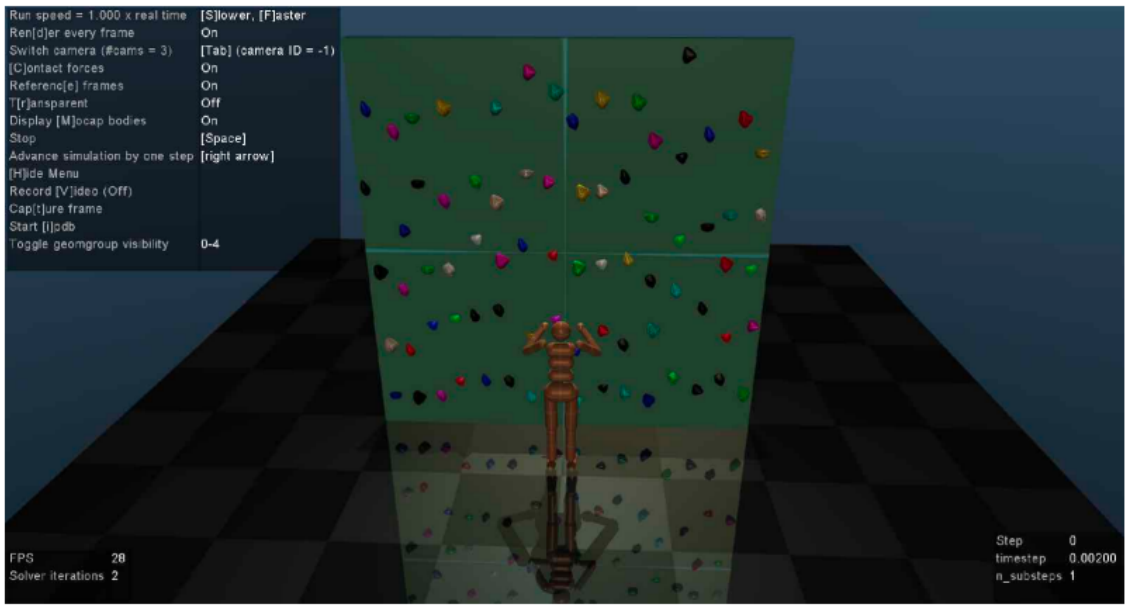


Figure 2.1 The simulation of the reinforcement learning model.

2008), propellers (Beardsley, Siegwart, Arigoni, Bischoff, Fuhrer, Krummenacher, Mammolo & Simpson, 2015), magnetic components (Eich & Vögele, 2011), or adhesive materials (Daltorio, Wei, Horchler, Southard, Wile, Quinn, Gorb & Ritzmann, 2009; Kim, Spenko, Trujillo, Heyneman, Mattoli & Cutkosky, 2007) to sustain their balance while they are climbing.

Another group of climbing robots has been designed and developed to climb around poles which are cylindrical surfaces, such as duct and pipe inspection. They can perform their actions by wrapping themselves around poles (Goldman & Hong, 2008) or squeezing their components to move (Haynes, Khripin, Lynch, Amory, Saunders, Rizzi & Koditschek, 2009).

Although all these climbing robots are bio-inspirational, none of them could perform free-climbing on vertical terrain.

To the best of the author's knowledge, *Lemur IIb* as shown in Figure 2.2, is the first free-climbing robot (Kennedy, Okon, Aghazarian, Badescu, Bao, Bar-Cohen, Chang, Dabiri, Garrett, Magnone & others, 2006); it has been built at Stanford University. *Lemur IIb* has four legs mounted on a circular chassis, and each leg possesses three degrees of freedom. Each end-effector, located at the tip of the legs, has a cylindrical "finger" covered with a high-friction rubber to increase the coefficient of

friction between the surface and the robot's contact point. In Bretl (2006), planning algorithms are presented for a two-link, three-link, and four-link (i.e., *Lemur IIb*) free-climbing robots. The planners in this work consist of two sequential stages. First, the planner computes a possible sequence of hand placements by finding feasible transitions. These sequences are computed after performing a set of feasibility checks, such as reachability, support region, and elbow bend checks. Having created a stance graph, path-finding is performed on this graph using a graph search, with path probability considered as a cost, which is inversely proportional to the amount of time spent exploring the stance and transition. In the second stage, the planner refines this sequence into a feasible continuous trajectory for the robot to follow by finding motion plans between subsequent transitions. For motion planning, the Probabilistic-Roadmap (PRM) (Kavraki et al., 1996b) approach is used to sample configurations between two states. Moreover, the feasibility of torque requirements of each motor on the robot to follow the motion plan in an open-loop manner is check via post-processing of the motion plans, as this feasibility check is time-consuming.

Figure 2.2 A free climbing robot, LEMUR.

A more simplified free-climbing robot, *Tenzing* as shown in Figure 2.3, has been built at Dartmouth College (Linder, Wei & Clay, 2005). Similar to *Lemur IIb*, it possesses four legs, each with two degrees of freedom. In addition to *Lemur IIb*, this robot is equipped with force sensors at the legs and a tilt sensor to measure the related variables. A camera located at some distance from the robot is used to determine the configuration of the robot. Similar to the planner of *Lemur IIb*, the planner of this robot utilizes an A* based global stance planner to determine

feasible pose transitions and a PRM-based local planner to find paths that connect these stances.

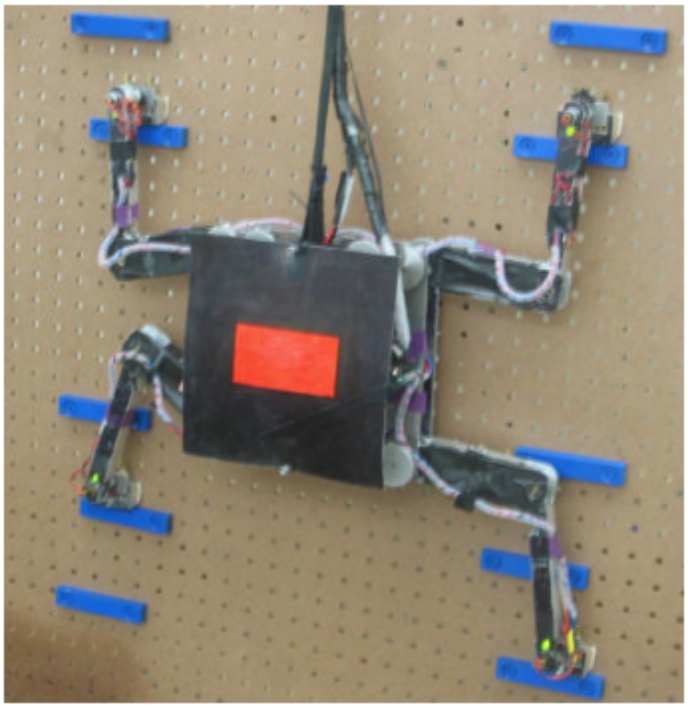


Figure 2.3 Tenzing robot at Dartmouth College.

Another free-climbing robot, called *Capuchin* (Zhang & Latombe, 2013) (see Figure 2.4), has been designed at Stanford University to improve upon *Lemur IIb*. *Capuchin* features an increased reachable workspace in comparison to *Lemur IIb*, thanks to addition of an extra degree of freedom at the pelvis of each leg, and increased joint limits. Furthermore, while *Lemur IIb* relied upon open-loop controllers, *Capuchin* utilizes feedback controllers to control the position and/or force at the tip of each leg (Hauser, 2008). *Capuchin* follows trajectories generated by a force-balancing controller (Miller, Bretl & Rock, 2006). The planner for *Capuchin* is very similar to that of *Lemur IIb*.

All of these previous works use similar methods to compute plans for a free climbing robot. They initially perform feasibility checks for stability and reachability and create a graph showing feasible transitions from one posture to another. Next, they use a graph search to perform path finding for a single robot from the initial posture to the final posture. After finding a discrete path as a sequence of postures, they use PRM-based motion planners to compute collision-free paths for each transition. Finally, the feasibility of actuator torques to follow these trajectories is validated.

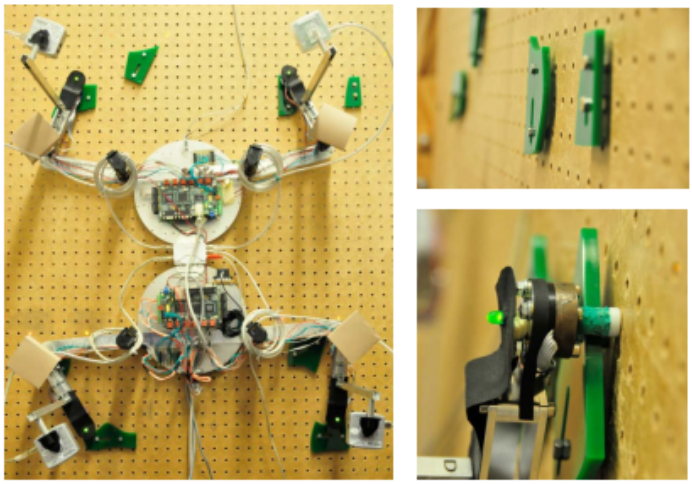


Figure 2.4 Capuchin on a climbing wall.

The procedure is sequential, and no-replanning loops are defined when a plan fails.

Similar to these works, our method first computes a feasibility graph considering balance and reachability constraints for the robot. Unlike other methods, our approach relies on a high-level reasoner to compute a sequence of poses on the feasibility graphs. To achieve more complex tasks, the high-level reasoner enables our approach to solve optimal plans with must-be-visited poses (waypoints) and must-be-avoided poses. Furthermore, our reasoner-based approach enables the easy extension of our method to accommodate multiple climbing robots. Once a sequence of poses are computed, our method also calls for a sampling-based motion planner to compute collision-free continuous trajectories that ensure robot balance. Unlike other methods, our motion planner is based on single-shot RRT-based methods and considers kinematic constraints due to the arms of the robots forming a closed-kinematic chain. This enables our motion planner to simultaneously move the body and arms of the robot to reach the desired pose, significantly improving the feasibility of motion plans.

Most importantly, our approach provides tight integration between high-level reasoning and low-level motion planning and feasibility checks. The high-level reasoner guides the motion planner to search for relevant pose transitions. The motion-planner and torque feasibility check guide the high-level reasoner via constraints that ensure computation of new plans that avoid pose transitions that are not feasible.

Table 2.1 presents a comparison of the related works with our method. We compare the type of planner used, feasibility analysis performed, and collision awareness. Also, we evaluate these methodologies in terms of the feasibility of actuator joint torques for the completeness of the robotic problem.

Reference	**Planner**	**Feasibility Checks**	**Obstacle Avoidance**	**Self-collision Avoidance**	**Actuator Torques Check**	**Waypoints**	**Multi-Robot Planning**
Bretl (2006)	PRM	Reachability, Stability	No	Yes	Yes	No	No
Zhang & Latombe (2013)	PRM	Reachability, Stability	No	Yes	Yes	No	No
Linder et al. (2005)	PRM	Reachability	No	Yes	No	No	No
This Thesis	**RRT-like**	**Reachability, Stability**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table 2.1 A comparison of the related work on free-climbing robots.

3. PRELIMINARIES

Our hybrid planning method for free-climbing multi-limbed robots utilizes Answer Set Programming for finding task plans, and Sampling-Based Motion Planning for finding continuous collision-free trajectories. Let us provide some preliminaries about them.

3.1 Answer Set Programming

Answer Set Programming (ASP)(Brewka et al., 2016; Lifschitz, 2002a; Marek & Truszczyński, 1999; Niemelä, 1999) is a declarative programming based on the stable model semantics of logic programming (Gelfond & Lifschitz, 1988,9; Lifschitz, 2002b). It is usually preferred to solve NP-hard search problems.

With ASP, we can define a computational problem as a finite set of **rules** (called a program P) whose answer sets correspond to solutions. The answer sets for a program P can be computed by ASP solvers, such as CLINGO (Gebser, Kaminski, Kaufmann & Schaub, 2014).

ASP programs consist of rules of the following form

$$\alpha \leftarrow \beta_1, \cdots, \beta_n, \texttt{ not } \beta_{n+1}, \cdots, \texttt{ not } \beta_m.$$

where $n \geq 0$ and $m \geq n$, each α is a propositional atom or $\bot$, and each β_i is

a propositional atom. In such a rule, α is called the **head** of the rule while $\beta_1, \cdots, \beta_n,$ `not` $\beta_{n+1}, \cdots,$ `not` β_m is the **body** of the rule. A rule where $\alpha = \bot$ is called a **(hard) constraint**; in such cases, since the head is empty, α is dropped from the rule. A rule where $n = m = 0$ is called a **fact**; in such a case, since the body is empty, the arrow is dropped from the rule.

In addition to hard constraints, ASP allows us to define *weak constraints*. They are expressions of the following form Buccafurri, Leone & Rullo (2000):

$$\widetilde{\leftarrow} \beta_1, \cdots, \beta_n, \texttt{ not } \beta_{n+1}, \cdots, \texttt{ not } \beta_m.[w@p, t_1, \cdots, t_n]$$

Intuitively, whenever an answer set for a program satisfies the body of a weak constraint, the tuple $\langle t_1, ..., t_n \rangle$ contributes a cost of w to the total cost function of priority p. Then the ASP solver tries to find an answer set with the minimum total cost. Different weak constraints might have different priority levels, and thus ASP can allow multi-objective optimization.

ASP allows other useful constructs used in this thesis: choice expressions, cardinality expressions, aggregates, optimization statements.

For instance, *cardinality expressions* are special constructs of the form $l\{A_1, \ldots, A_k\}u$ where each A_i is an atom and l and u are nonnegative integers denoting the lower and upper bounds Simons, Niemelae & Soininen (2002). Such an expression describes the subsets of the set $\{A_1, \ldots, A_k\}$ whose cardinalities are at least l and at most u. Cardinality expressions can be used in the heads of rules; then they generate many answer sets whose cardinality is at least l and at most u. For instance, the answer sets for the program $2\{p_1, ..., p_7\}4.$ are subsets of the set $\{p_1, ..., p_7\}$ whose cardinality is at least 2 and at most 4. Such rules with a cardinality expression in the head are called *choice rules*.

We present an ASP program to an ASP solver, with slight changes in the syntax. For instance, in the input language of CLINGO, `:-` stands for $\leftarrow$, and each rule is followed by a period.

An example. Consider the Traveling Salesperson Problem. A salesperson aims to visit a given set of cities with a shortest route, so that each city is visited exactly ones; the salesperson starts and ends the route in the same city. We represent this problem in ASP and present it to CLINGO as described in Gebser, Kaufmann, Kaminski, Ostrowski, Schaub & Schneider (2011) as follows:

```
1  % Generate
2  { cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
3  { cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).
4  % Define
5  reached(Y) :- cycle(1,Y).
6  reached(Y) :- cycle(X,Y), reached(X).
7  % Test
8  :- node(Y), not reached(Y).
9  % Display
10 #show cycle/2.
11
12 % Optimize
13 #minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

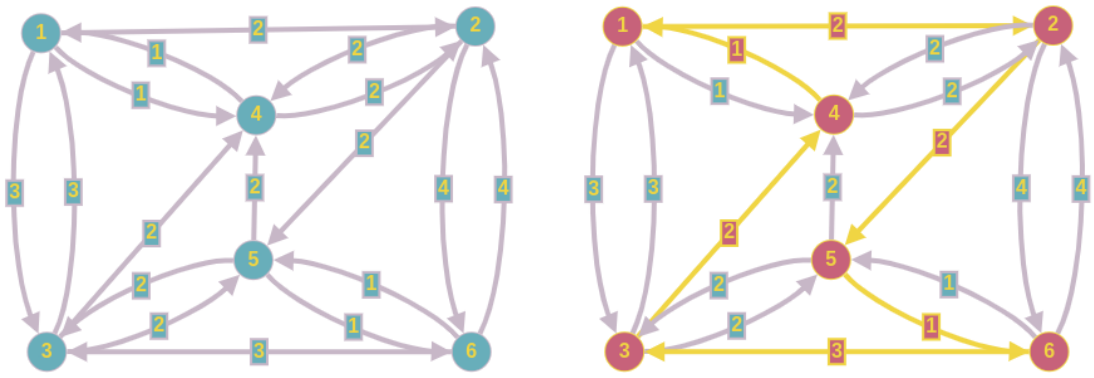
This ASP program has five pars: generate (lines 2, 3), define (lines 5, 6), test (line 8), display (line 10) and optimize (line 13). In lines 2 and 3, a subgraph is generated including one edge to and one edge from every node. Note that this subgraph is a cycle: Every node in a cycle has exactly one incoming and one outgoing edge. In lines 5 and 6, the reachability of a node from node 1 is defined. Line 8 eliminates cycles when a node is not reachable. Line 10 displays the cycle. Line 13 minimizes the total cost of the edges included in the cycle.

This ASP program assumes that a problem instance is provided as a graph, where the nodes denote cities and the edges denote the roads. Here is an example in the input language of CLINGO:

```
1 % Nodes
2 node(1..6).
3 % (Directed) Edges
```

```
4  edge(1,(2;3;4)).   edge(2,(4;5;6)).   edge(3,(1;4;5)).
5  edge(4,(1;2)).     edge(5,(3;4;6)).   edge(6,(2;3;5)).
6
7  % Edge Costs
8  cost(1,2,2).   cost(1,3,3).   cost(1,4,1).
9  cost(2,4,2).   cost(2,5,2).   cost(2,6,4).
10 cost(3,1,3).   cost(3,4,2).   cost(3,5,2).
11 cost(4,1,1).   cost(4,2,2).
12 cost(5,3,2).   cost(5,4,2).   cost(5,6,1).
13 cost(6,2,4).   cost(6,3,3).   cost(6,5,1).
```

According to the instance, there are six nodes that are connected through directed edges. Each directed edge has a unique cost value. The schematic representation of this instance is visualised on Fig. 3.1



(a) The problem instance for the Traveler Salesperson Problem.

(b) The optimum solution the problem.

Figure 3.1 Visualisation for the Traveler Salesperson Problem instance

A solution to this problem instance can be found by CLINGO as follows:

```
Answer: 1
cycle(1,4) cycle(4,2) cycle(3,1) cycle(2,6) cycle(6,5) cycle(5,3)
Optimization: 13
Answer: 2
cycle(1,4) cycle(4,2) cycle(3,1) cycle(2,5) cycle(6,3) cycle(5,6)
Optimization: 12
Answer: 3
```

```
cycle(1,2) cycle(4,1) cycle(3,4) cycle(2,5) cycle(6,3) cycle(5,6)
Optimization: 11
OPTIMUM FOUND

Models       : 3
  Optimum    : yes
Optimization : 11
Calls        : 1
Time         : 0.213s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.007s
```

This output describes a shortest cycle that the salesperson can follow to visit every node. The salesperson starts and ends the route at node 1.

3.2 Sampling-Based Motion Planning

In this thesis, the proposed approach relies on sampling-based planning with geometric constraints (LaValle, 2006). Geometric constraints are required in many applications, such as during manipulation tasks (Siméon, Laumond, Cortés & Sahbani, 2004), for parallel manipulators and robots forming closed kinematic chains (LaValle, 2006; Tenenbaum, De Silva & Langford, 2000).

There are two main categories of sampling-based planners, which are multi-shot graph-based methods, such as PRM (Kavraki, Svestka, Latombe & Overmars, 1996a), and single-shot tree-based methods, such as RRT (LaValle & James J. Kuffner, 2001). Graph-based methods build a "roadmap" in the free configuration space, while tree-based methods construct a tree of feasible configurations starting from the start or the goal configuration, or both, as bidirectional trees have been demonstrated to improve the computational efficiency of motion planning.

A configuration q of a robot is defined as $(q \in Q)$, where Q is the *configuration space*, or the metric space. The degree-of-freedom of a robot is represented by a positive number, n. The basic motion planning problem is defined as finding a continuous collision-free path from q_{start} to q_{goal} in the configuration space, such that $\tau : [0,1] \rightarrow Q$, $\tau(0) = q_{start}$, and $\tau(1) = q_{goal}$. For the constrained motion planning, we also want to satisfy a *constraint function* $F(q) = 0$ such that every feasible q satisfies the constraint. Consider there are k number of equality constraints and these constraint functions define an $(n-k)$-dimensional implicit constrained configuration space in the ambient configuration space.

$$\mathcal{X} = \{q \in Q | F(q) = 0\}$$

In this work, we consider the constraint functions F that are continuous and differentiable everywhere, so $\mathcal{X}$ is a manifold. The constrained motion planning problem with a constraint function F and configuration space Q is a problem of finding

$\tau : [0,1] \rightarrow \mathcal{X}$. In this study, the constrained planning module of the Open Motion Planning Library (OMPL) is used (Kingston, Moll & Kavraki, 2019).

There are three representations of constrained spaces in the OMPL. They contain different implementations for sampling and traversing the problem. The first one is based on Projected State Space that uses a projection operation to find satisfying constraint motion. The second approach is the Atlas State Space method that builds piecewise linear approximation of constraint functions while planning. The last approach is the Tangent Bundle State Space, similar to the Atlas, but it lazily evaluates the approximations. Considering the critical nature of constraint equations of a free climbing robot, we use Projected State Space in this study.

4. HYBRID PLANNING FOR FREE CLIMBING ROBOTS

Let us first define the problem addressed by our hybrid planning method, and provide an outline of our method.

4.1 HFCP: Hybrid free climbing Planning Problem

The hybrid free climbing planning problem (called HFCP) for a multi-limbed robot is characterized by the following input:

- a climbing environment $\mathcal{E}$ consisting of sets of H number of holds $\mathcal{H} = \{0, 1, \ldots, H-1\}$ and K number of obstacles $\mathcal{O} = \{0, 1, \ldots, K-1\}$,
- a multi-limbed robot kinematics/dynamics $\mathcal{R}$ having N number of legs,
- an initial state of the robot as an $N-$tuple of holds $\sigma_i = (s_0, s_1, \ldots, s_{N-1})$ such that $s_j \neq s_k$, $\forall s_j, s_k \in \mathcal{H}$ where $0 \leq j, k \leq N-1$,
- an initial configuration of the robot $\mathbf{q}_i$ satisfying the initial state σ_i,
- a final region for the pelvis of the robot, R_p^f,
- a set of must-be-visited pelvis regions, $\mathcal{W} = \{R_p^0, R_p^1, \ldots, R_p^l\}$.

A solution for an HCFP instance $(\mathcal{H}, \mathcal{O}, \mathcal{R}, \sigma_i, \mathbf{q}_i, R_p^f, \mathcal{W})$ is a *feasible free-climb plan* for a multi-limbed robot on the environment $\mathcal{E}$.

We compute a solution to HCFP by means of generating a *feasibility graph*, whose

nodes denote stable poses of the robot, the edges denote reachable moves of the robot (by moving only one of its legs), and the costs of edges characterize an aggregate cost of stability, reachability and robustness.

Let us outline the steps of our hybrid planning method to solve HCFP:

1.1 Considering the given climbing environment, a starting initial state, and a final position of the body of the robot, first a feasible set of states are generated. A state represents which leg is contacting which hold. Each state is denoted by a node in the feasibility graph.

1.2 Then, starting from the given initial state, the adjacent states that can be reached by the robot (where only one of its legs and its body are moved) are identified. A state and its adjacent state are denoted by an edge in the feasibility graph.

1.3 Next, a cost of the edge is defined considering stability, reachability and robustness.

1.4 After that, we compute a task plan over this feasibility graph, to decide for the moves of the robot where the states and transitions are feasible in terms of reachability and stability.

1.5 Then, we utilize motion planning to check whether each move of the robot is collision-free.

1.6 Finally, we use an inverse dynamics controller to verify feasibility of actuator torques required to follow the computed trajectories.

Let us describe each step of our method in detail.

4.2 Generating a Feasibility Graph

A *feasibility graph* is a weighted bidirected graph. It consists of nodes that denote stable poses of the robot, and edges that denote reachable moves of the robot (by moving only one of its legs). The weights of edges characterize an aggregate cost of stability, reachability and robustness.

4.2.1 State Creator (Vertex Creator)

A *state* is an $N-$tuple of holds, where N is the number of legs of the robot. In fact, it is a unique posture for the free climbing robot, and it contains the information of which leg of the robot is attached on which hold, respectively. For example, a state, $\sigma = (5, 1, \ldots, 0)$, defines that the first leg is attached on the hold number 5, the second leg is attached on the hold number 1, and the N^{th} leg is attached on the hold number 0. Note that each leg of the robot is allowed to connect only one hold at each state. Therefore, $s_i \neq s_j, \forall s_i, s_j \in \mathcal{H}$ in a state $\sigma = (s_0, s_1, \ldots, s_{N-1})$.

For the hybrid planning problem, the first step is to find all feasible states in the given environment $\mathcal{E}$ because they will be used to define the stance of the robot and to find the related configuration parameters. In order to find feasible states, two low-level checks are performed after finding all $N-$tuple of holds using the permutation operation of all holds. As presented in Eqn. (4.1), $n(\Sigma')$ which is the number of the set of all $N-$tuple of holds in $\mathcal{H}$ are calculated using the permutation operation on all number of holds.

$$n(\Sigma') = {}^{H}P_N = \frac{H!}{(H-N)!} \tag{4.1}$$

In a climbing environment having H holds, there exists $H!/(H-N)!$ states, and each of them needs to be validated in terms of feasibility. There are two different

low-level checks are used for this purpose. The first one, called the State Validator (see Section 4.2.1.1), checks if the given state is suitable for the robot to have a kinematic placement. Second one is to prevent leg crossings (see Section 4.2.1.2) while the robot is performing a state.

4.2.1.1 State Validator

For the set of states found after the permutation operation in Eqn. (4.1), there can be some situations where holds are too far away from each other in a state such that the robot cannot be physically placed. In order to eliminate these states in Σ', we apply a low-level check named *State Validator*.

State Validator considers the free climbing robot as if it is a parallel mechanism, and each leg of the robot is attached at the center of each hold. The aim is to find if there is a maximal workspace for the assumed parallel mechanism by performing a reachability check using circles (Merlet, Gosselin & Mouly, 1998). We draw N circles around each hold for a state, $\sigma' \in \Sigma'$, and their radius equal to the maximal leg length of the robot, i.e. it is the maximum distance from the center of the body to the tip of a leg. Also, center points of these circles are located at the center points of each hold in the state.

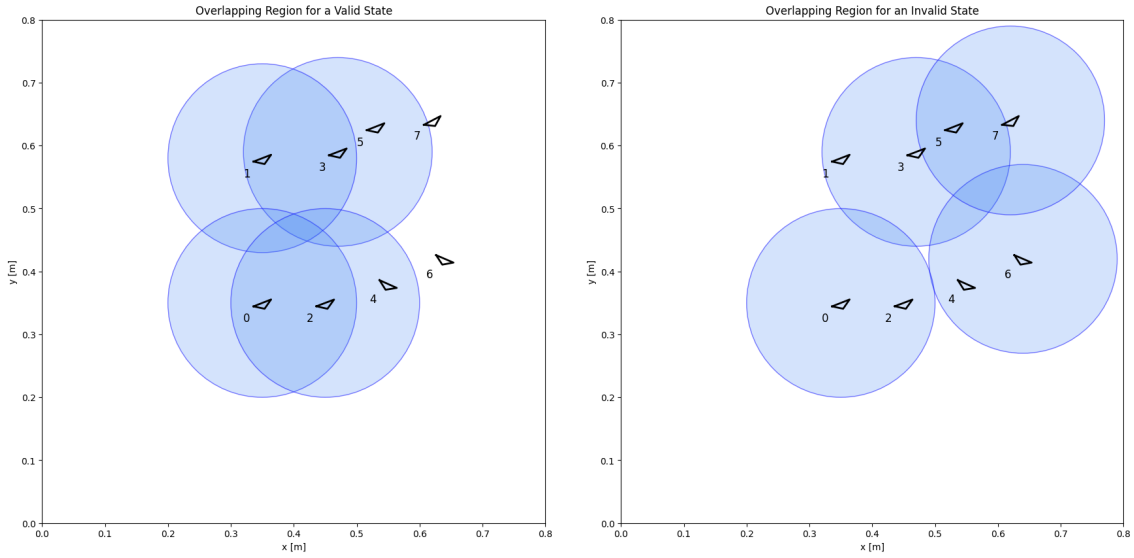
State Validator simply checks if there is an overlapping region for the circles drawn. If the region exists, this state is marked as a valid state, and it is not a valid state otherwise. In other words, if all circles are intersecting each other, then there is a kinematic solution for the assumed parallel mechanism, and we can place the free climbing robot while its legs are attached the holds in that state.

Let $C(i)$ be a circle drawn around the hold number i with the radius equals to equal to the maximal leg length of the robot. A valid state exists if all circles in a state

create an overlapping region.

$$\sigma' \text{ is } \begin{cases} \text{a valid state,} & C(i) \cap C(j) \cap \ldots \cap C(k) \neq \emptyset \\ \text{not a valid state,} & \text{otherwise} \end{cases} \tag{4.2}$$

Fig. 4.1 explains Eqn. (4.2) visually. For a robot having four legs, i.e. $N = 4$, there are four circles drawn for two candidate states. In Fig. 4.1a, there is an overlapping region of the circles drawn around holds: 0, 1, 2, 3 for $\sigma' = (0,1,2,3)$, so this state is a valid state, meaning that there is a kinematic solution to place the robot whose legs are connected to these holds. On the other hand, there is a infeasible state in Fig. 4.1b because there is not an overlapping region of circles drawn around holds: 0, 3, 6, 7 for $\sigma' = (0,3,6,7)$. Therefore, we eliminate these states in Σ', and move to the next low-level check to finalize the valid state definitions.



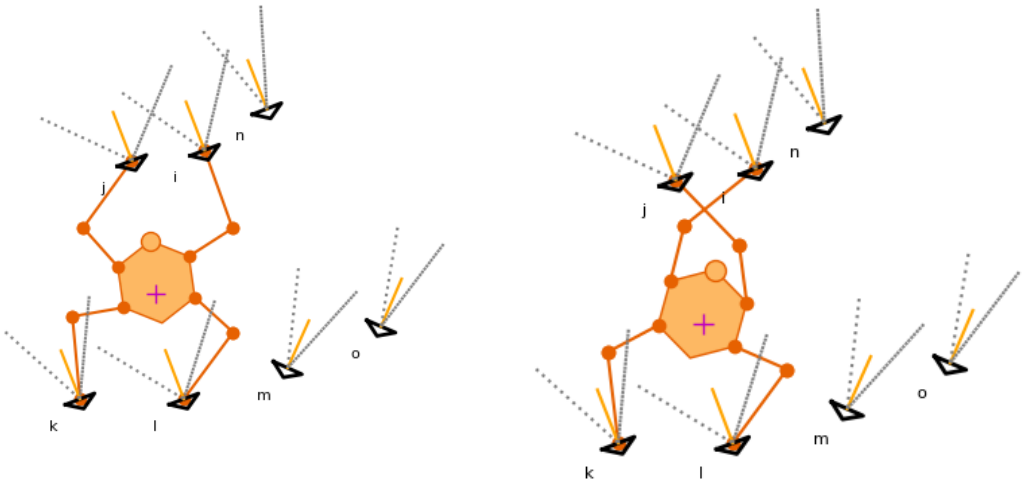
(a) An example of valid state, $\sigma' = (0,1,2,3)$ due to the overlapping region by circles drawn around holds number: 0, 1, 2, 3

(b) A sample of invalid state, $\sigma' = (0,3,6,7)$ as circles don't create an overlapping region by holds number: 0, 3, 6, 7

Figure 4.1 A sample evaluation of the State Validator

4.2.1.2 Leg Crossing Check

Although State Validator eliminates all infeasible states where the robot cannot be placed due to reachability, it does not take the leg crossings into account. Leg crossing can cause the robot to self-intersect, resulting in an infeasible configuration. Due to the order-dependent nature of states (as the state definition is in $N-$tuple), we need to consider the placement of each leg. For example, the case where a leg on the right side of the body is attached to hold at the left side of the body, and a leg on the left is connected to hold at the right may have a self-intersection between these legs, as presented in Fig 4.2. In Fig 4.2a, there is a state, $\sigma' = (i, j, k, l)$, where there is no leg crossing, but in Fig 4.2b, there is an infeasible state for $\sigma' = (j, i, k, l)$ with leg crossing. Therefore, two states containing the same holds may cause infeasibility due to leg crossings of the robot.



(a) An example of state with not crossing legs, $\sigma' = (i, j, k, l)$.

(b) An example of state with crossing legs, $\sigma' = (j, i, k, l)$.

Figure 4.2 A sample case for the leg crossing.

The leg-crossing is checked by studying the angles of two vectors that are defined by center points of sequential holds. For example, if we consider a free climbing robot having four legs, i.e. $N = 4$, and a state given as $\sigma' = (i, j, k, l)$, meaning the first leg is attached to the i^{th} hold, the second one is on the j^{th} hold, and so on. In order to check the leg crossing for each leg, we check the argument of two vectors drawn

from a hold.

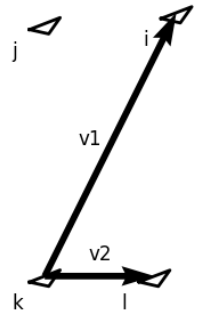


Figure 4.3 A step for the Leg Crossing Check.

First, we draw a vector $\mathbf{v}_1$ from k^{th} hold to i^{th} hold. The argument of $\mathbf{v}_1$ is proportional to the $\tan\alpha_1$, where α_1 is the angle between the $\mathbf{v}_1$ and our reference frame that is the horizontal axis. Similarly, we draw the consequent vector $\mathbf{v}_2$ from k^{th} hold to l^{th} hold, and its argument is proportional to the $\tan\alpha_2$ as stated in Eqn. (4.3).

$$\begin{aligned} \arg(\mathbf{v}_1) \propto \tan\alpha_1 &= \frac{y_i - y_k}{x_i - x_k} \\ \arg(\mathbf{v}_2) \propto \tan\alpha_2 &= \frac{y_l - y_k}{x_l - x_k} \end{aligned} \tag{4.3}$$

Leg crossing can be avoided when the argument of the first vector is greater than the argument of the second vector:

$$\begin{aligned} &\arg(\mathbf{v}_1) > \arg(\mathbf{v}_2) \\ &\frac{y_i - y_k}{x_i - x_k} > \frac{y_l - y_k}{x_l - x_k} \end{aligned} \tag{4.4}$$

Simplifying Eqn. (4.4),

$$(y_i - y_k)(x_l - x_k) - (x_i - x_k)(y_l - y_k) > 0 \tag{4.5}$$

Other three checks can be preformed in similar way, and the following equations can be obtained. If all of equations are satisfied, we can conclude that the state has no leg crossings.

$$\begin{aligned}
(y_i - y_k)(x_l - x_k) - (x_i - x_k)(y_l - y_k) > 0 \\
(y_j - y_k)(x_i - x_k) - (x_j - x_k)(y_i - y_k) > 0 \\
(y_j - y_l)(x_i - x_l) - (x_j - x_l)(y_i - y_l) > 0 \\
(y_k - y_l)(x_j - x_l) - (x_k - x_l)(y_j - y_l) > 0
\end{aligned} \tag{4.6}$$

4.2.2 Edge Creator

Having a set of feasible states in the climbing environment, $\mathcal{E}$, we need to consider feasible transitions between them, such that a feasibility graph can be constructed. The main assumption for the transition from a state to another is that the robot can move only one leg while the others keep their positions during the motion. This is needed because the motion is quasi-static motion, so the equations of static equilibrium are valid. Therefore, starting from the initial state σ_i, we check all states that are near to the initial state in terms of feasibility analysis. There are two checks which are Reachability Check (see Section 4.2.2.1) to evaluate the robot can reach from a valid state to another valid state, and Stability Check (see Section 4.2.2.2) to be sure that the transition satisfies equations of static equilibrium, so the robot can move without falling.

4.2.2.1 Reachability Check

The purpose of the Reachability Check is to evaluate whether a state is reachable from another state, or not. If the hold is reachable, we can consider that there is a transition from the current state σ_c to a new state σ_n including the reachable hold, but excluding the released hold.

Reachability Check has similar procedure as described in Section 4.2.1.1. However, the concern is not only to find a workspace, but also to find if there is an intersection between the workspace of the robot and the circle around the candidate hold. First, the free climbing robot having N legs is considered as a planar parallel manipulator. After, the maximal workspace of the planar mechanism is investigated by drawing circles centered at the middle points of each hold. The radius of these circles are all the same, due to the wing length of the robot, which is the maximum distance from the center of the body to the tip of a leg. The overlapping region of these N circles is the maximal workspace, and if the circle drawn around the desired hold is

intersecting with the overlapping region, we can say this hold is reachable.

For instance as presented in Fig. 4.4, the aim is to check the reachability form $\sigma_c = (i, j, k, l)$ to $\sigma_n = (i, j, k, m)$. In other words, we check the reachablity of the right rear leg of the robot from the hold l to the hold m. To do that, we check the intersection of two workspaces of the two states, and as shown in the figure, it is reachable.

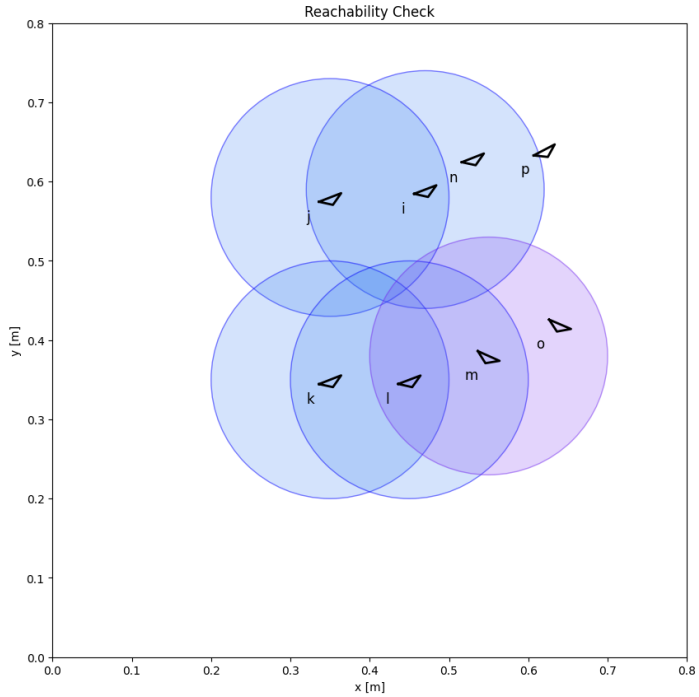


Figure 4.4 Reachability Check from a state to a hold

Consider a current state $\sigma_c = (i, j, ..., k)$, a new state $\sigma_n = (l, m, ..., n)$ and a free climbing robot having N legs with a maximal leg length of r. Draw circles around center points of each hold of states with radius of r, and let $C(i)$ be a circle drawn around the hold number i. As formulated in (4.7), the state σ_n is reachable by the state σ_c if all circles in these states create an overlapping region.

Let the workspace of the current state be $W_c = C(i) \cap C(j) \cap ... \cap C(k)$, then

$$\sigma_n \text{ is } \begin{cases} \text{reachable by } \sigma_c, & W_c \cap C(l) \cap C(m) \cap ... \cap C(n) \neq \emptyset \\ \text{not reachable by } \sigma_c, & \text{otherwise} \end{cases} \tag{4.7}$$

4.2.2.2 Stability Check

Consider a climbing environment designed as a collection of various pegs, such as the rock-climbing pegs on a climbing wall at gyms. These pegs are called *holds*, and they are assumed to lie in a single patch, so each hold has a normal vector $\hat{\mathbf{v}}_i$. Then, considering dry Coulomb friction, the reaction force $\mathbf{f}_i$ at each hold i is constrained to be in a circular (quadratic) friction cone FC_i of the static coefficient of friction μ_i centered about the normal vector $\hat{\mathbf{v}}_i$.

For simplicity, a triangular approximation to FC_i is used on the plane. However, in space, this approximation can be applied as a p-gonal pyramid, and increasing p increases both the precision of FC_i and the computational cost (Bretl, 2005). As we work on a plane, we take $p = 2$, which makes the friction cone to assume a triangular region.

Define unit vectors $\hat{\mathbf{f}}_{i1}$ and $\hat{\mathbf{f}}_{i2}$ along each edge of the triangle. The reaction force

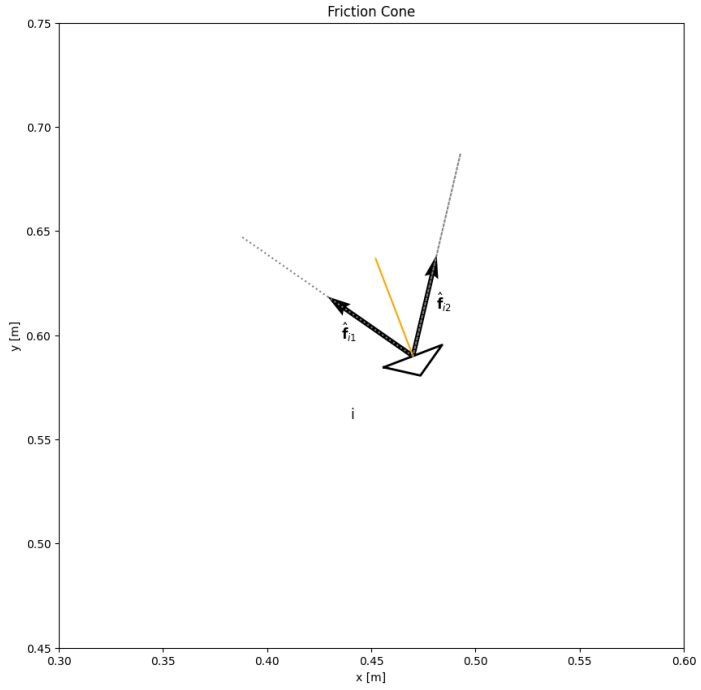


Figure 4.5 Friction Cone on a hold

$$\mathbf{f}_i = \sum_{j=1}^{p=2} f_{ij}\hat{\mathbf{f}}_i \tag{4.8}$$

lies with the triangle approximating FC_i if $f_{i1}, f_{i2} \geq 0$. In practice, this constraint is defined as a minimum and a maximum limit.

$$f_{min_i} \leq f_{ij} \leq f_{max_i} \text{ for } (j = 1, 2). \tag{4.9}$$

The vector of variables f_{ij} will be denoted as

$$\mathbf{f}_i = \begin{bmatrix} f_{i1} \\ f_{i2} \end{bmatrix} \tag{4.10}$$

so this constraint can be expressed as follows ("$\preceq$" denotes vector inequality):

$$f_{min_i}\mathbf{1} \preceq \mathbf{f}_i \preceq f_{max_i}\mathbf{1} \tag{4.11}$$

Since we assume the motion of the free climbing robot is quasi static, we need to consider equations of static equilibrium to compensate for gravity. To do this, let $\mathbf{r_i} = (x_i, y_i)$ be the location of each hold i on the plane. Assume the robot has a mass m, and the center of mass (CM) is located at (x_{CM}, y_{CM}) with respect to a fixed coordinate frame, and gravity is $\mathbf{g} = (0, -g)$. Define following matrix for each hold:

$$\mathbf{F}_i = \begin{bmatrix} \hat{\mathbf{f}}_{i1} & \hat{\mathbf{f}}_{i2} \\ \mathbf{r}_i \times \hat{\mathbf{f}}_{i1} & \mathbf{r}_i \times \hat{\mathbf{f}}_{i2} \end{bmatrix} \tag{4.12}$$

Then, the free climbing robot is in static equilibrium in both force and torque if the reaction forces exist such that

$$\begin{bmatrix} & & 0 \\ \mathbf{F}_1 \cdots \mathbf{F}_n & & 0 \\ & & -mg \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_n \\ x_{CM} \end{bmatrix} = \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix} \tag{4.13}$$

The first two lines of Eqn. (4.13) capture the static force balance in a fixed coordinate

frame, respectively. Summation of reaction forces along the horizontal direction should be equal to zero. Similarly, summation of reaction forces along the vertical direction should be equal to the weight of the robot such that the static equilibrium is satisfied.

The constraints Eqn. (4.11) and Eqn. (4.13) can be represented in a general form to be solved together. First, let

$$\mathbf{x} = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_n \\ x_{CM} \end{bmatrix}$$

so $\mathbf{x} \in \mathbb{R}^{2n+1}$. Then, define

$$\mathbf{A}_{eq} = \begin{bmatrix} & & 0 \\ \mathbf{F}_1 \cdots \mathbf{F}_n & & 0 \\ & & -mg \end{bmatrix}_{3 \times 2n+1} \quad \text{and } \mathbf{b}_{eq} = \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix}$$

Then, Eqn. (4.13) can be represented as:

$$\mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \tag{4.14}$$

Similarly, define $\mathbf{x}' \in \mathbb{R}^{2n}$ to be used in the boundary constraint,

$$\mathbf{x}' = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_n \end{bmatrix}$$

Then, the minimum and the maximum boundaries can be defined in the following

form,

$$\mathbf{x}'_{min} = \begin{bmatrix} f_{min_1}\mathbf{1} \\ \vdots \\ f_{min_n}\mathbf{1} \end{bmatrix} \text{ and } \mathbf{x}'_{max} = \begin{bmatrix} f_{max_1}\mathbf{1} \\ \vdots \\ f_{max_n}\mathbf{1} \end{bmatrix}$$

so from Eqn. (4.11), $\mathbf{x}$ is bounded by

$$\mathbf{x}'_{min} \preceq \mathbf{x}' \preceq \mathbf{x}'_{max} \tag{4.15}$$

Then, define

$$\mathbf{A}_{ub} = \begin{bmatrix} -\mathbf{I} & \mathbf{0}_{4n\times 1} \\ \mathbf{I} & \end{bmatrix}_{4n\times 2n+1} \text{ and } \mathbf{b}_{ub} = \begin{bmatrix} -\mathbf{x}'_{min} \\ \mathbf{x}'_{max} \end{bmatrix}_{4n\times 1}$$

Note that, the last raw of $\mathbf{A}_{ub}$ is a zero array corresponding to x_{CM} in $\mathbf{x}$, which is unbounded and our aim is to find a feasible boundary for x_{CM}. Therefore, Eqn. (4.11) can be represented as

$$\mathbf{A}_{ub}\mathbf{x} \leq \mathbf{b}_{ub} \tag{4.16}$$

Now, we have a linear equality in Eqn. (4.14) and an inequality constraints Eqn. (4.16), and our target is to find a range for x_{CM} satisfying these two constraints, which can be solved by *Linear Programming*. Linear Programming solves problems of the following form:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{such that} \quad & \mathbf{A}_{ub}\mathbf{x} \leq \mathbf{b}_{ub} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \end{aligned}$$

where $\mathbf{x}$ is a vector of decision variables, $\mathbf{c}, \mathbf{b}_{ub}$ and $\mathbf{b}_{eq}$ are vectors, and $\mathbf{A}_{ub}$ and $\mathbf{A}_{eq}$ are matrices. Since the Linear Programming optimizes $\mathbf{c}^T\mathbf{x}$, we need to choose

a proper $\mathbf{c}$ vector such that it gives the minimum and the maximum values for x_{CM}. Therefore,

$$\mathbf{c}_{min} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \text{ to find the minimum } x_{CM}$$

and

$$\mathbf{c}_{max} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \text{ to find the maximum } x_{CM}$$

The results of the Linear Programming for both the minimum and the maximum x_{CM} defines a region where the center of mass of the robot should lie. In Fig. 4.6, the darker region is a support region for a state when the robot is moving its right front leg, and the area including both lighter and darker region is a support region when all four legs of the robot are on hold. The intersection region of these two support regions enables the robot to move without falling.

Let a current state and a next state be $\sigma_c = (i, j, k, l)$ and $\sigma_n = (n, j, k, l)$, the stability check considers the support region while the robot is performing an action. Since the first leg is moving, the current support region SR_1 is a function of (j, k, l) because these holds keep the robot in balance. Similarly, to see if there is a stable transition, the support region for the next state SR_2 that is supported by (n, j, k, l) should intersecting with SR_1.

$$\text{The transition is } \begin{cases} \text{stable} , & SR_1 \cap SR_2 \neq \emptyset \\ \text{not stable} , & \text{otherwise} \end{cases} \tag{4.17}$$

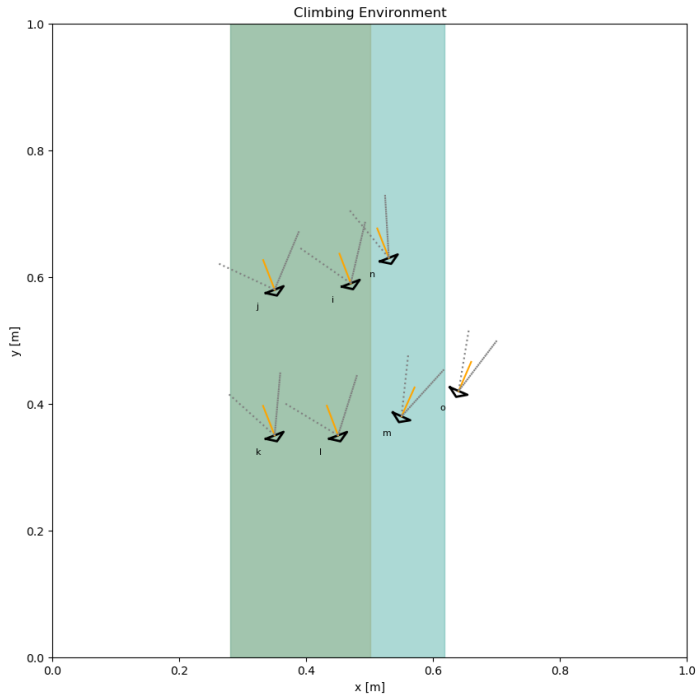


Figure 4.6 An example of Stability Check

4.2.3 Constructing the Feasibility Graph

One of the main aspects of the hybrid planning is to convert the problem into a graph planning problem based on the feasibility checks explained in the earlier sections. For the given climbing environment, a starting initial state, and a final position of the body of the robot, the method creates a graph to be used for planning. First, it finds feasible sets of holds, *states*. A state has N number of holds representing which leg is contacting at which hold. Second, starting from the given initial state, the adjacent states which have only one different hold than the initial state in the set are checked, as described in Section 4.2.2. For example, consider the following set of holds and the initial state for $N = 4$ in a given environment,

$$\mathcal{H} = \{0, 1, ..., 9\}$$

$$\sigma_i = (3, 1, 0, 2)$$

The initial state describes the hold 3 is contacting with the first leg, hold 2 is

contacting with the second leg, etc. An adjacent state is the state that only one hold in σ_i is changed because the robot is allowed to move only one of its leg at each time step, due to the quasi-static motion assumption. The following states include the list of states that are checked by feasibility analysis:

$$(4,1,0,2),(5,1,0,2),\ldots,(9,1,0,2)$$
$$(3,4,0,2),(3,5,0,2),\ldots,(3,9,0,2)$$
$$(3,1,4,2),(3,1,5,2),\ldots,(3,1,9,2)$$
$$(3,1,0,4),(3,1,0,5),\ldots,(3,1,0,9)$$

Therefore, for four legs ($N = 4$) and 10 holds in a climbing environment ($H = 10$), there are 24 possible states from an initial state whose feasibility need to be checked. The general formula to compute the number of possible states from a starting state is given in Eqn. (4.18), where N is number of legs of the free climbing robot, and K is the number of holds in the climbing environment.

$$\text{Number of Possible States} = N(K-N) \tag{4.18}$$

Each adjacent state is checked and if it is a feasible state, it is saved. Unless states are in this saved file, they are not considered anymore as a node in the graph. On the other hand, the feasibility of states are checked as described in Section 4.2.2. If we find there is a feasible transition, we add feasible nodes and their edges to the graph. This procedure is performed to cover all states in the environment.

The edges are assigned same weights to find the path having the least cost. We define some cost parameters for weighted edges. The visualization for an edge is presented in Fig. 4.7. This edge is defined from $\sigma_1 = (i,j,k,l)$ to $\sigma_2 = (n,j,k,l)$. The parameters on the figures are d_{reach}, which is the Euclidean distance from the previous hold to the next hold for the active leg, d_{stab} that is the width of the stable region defined on supportive legs, i.e. for holds at (j,k,l), and A_{robust} that is the maximal workspace of the robot while the active leg is free.

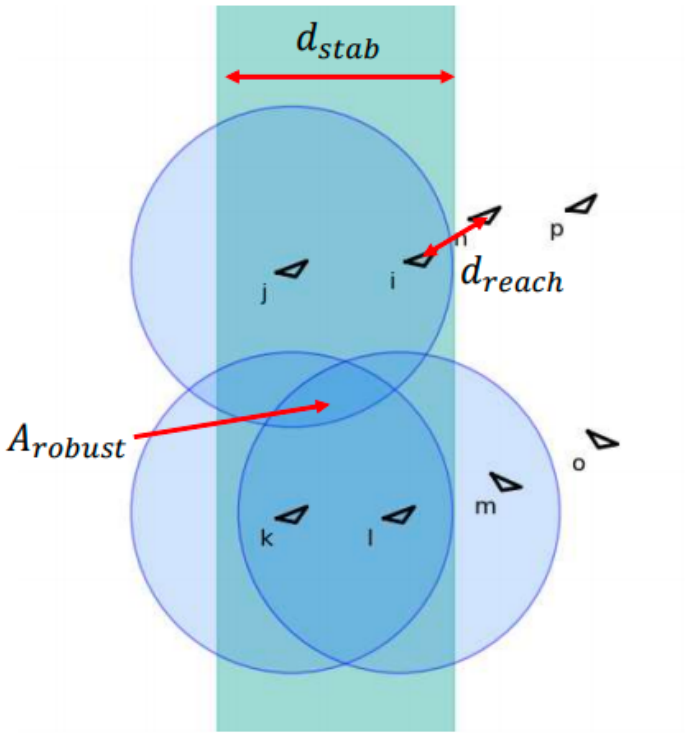


Figure 4.7 Cost definition on edges.

4.2.3.1 Reachability Cost

Reachability cost is defined as the Euclidean distance from the current hold to the next hold for the active leg. As the robot is allowed to move only one of its leg at each step, there is an active leg that moves while the others keep their positions. The distance from the current hold to the next hold is defined as reachability cost, and it is proportional to this distance measure.

$$C_{reach} = d_{reach} \tag{4.19}$$

where d_{reach} is the distance from the current hold to the next hold for the active leg. Note that the cost is defined after the feasibility analysis.

4.2.3.2 Stability Cost

Stability cost is defined through the distance between the maximum and the minimum positions of the support polygon while the active leg is not contacting to any point. $N-1$ legs of the robot is contacting to their holds, and they are responsible to balance the robot when the active leg is moving. Therefore, the center of mass of the robot should lie in the support region caused by $N-1$ legs as described in Section 4.2.2.2. Note that, this cost is inversely proportional to the distance because larger stable region enables more freedom to the robot, so we invert the distance as in Eqn. (4.20).

$$C_{stab} = \frac{1}{d_{stab}} \tag{4.20}$$

where d_{stab} is the width of the support region caused by $N-1$ legs, i.e. $\|x_{CM_{\max}} - x_{CM_{\min}}\|$.

4.2.3.3 Robustness Cost

Robustness cost is computed based on the area where the body of the robot can be placed during the transition from a state to the other. This area is the area of the workspace that is the overlapping region of intersecting $N-1$ circles centered at each hold, and it is calculated using Shapely Library (Gillies, 2020). Since a larger area enables more freedom for the robot to move, the cost is inversely proportional to the area.

$$C_{robust} = \frac{1}{A_{robust}} \tag{4.21}$$

4.2.3.4 Weighted Cost

Weighted cost is the weighted sum of all costs to aggregate all costs into a single cost.

$$C = W_1 C_{reach} + W_2 C_{stab} + W_3 C_{robust} \tag{4.22}$$

4.3 Task Planning using ASP

Having a weighted graph, a starting node and a final node, we can find the shortest least-cost plan using ASP with a multi-shot computation (Gebser et al., 2019).

Symmetry breaking. As described in Section 4.2, each state is an $N-$tuple of holds in the environment. The size of states is N, that is, the number of legs of the free climbing robot, and the order of holds in a state is order dependent representing which leg is contacting with which hold. Considering the symmetric structure of the robot, the order dependency can be neglected, because at the high-level, they are represented by the same configuration of the robot. This enables us to have more compact graph having less edges and nodes, so improves the computational performance of task planning. Therefore, we convert an order dependent state to a unique state such that hold numbers are sorted. For example, $\sigma = (4, 1, 0, 2)$ is changed to the $\sigma' = (0, 1, 2, 4)$ such that hold indices are sorted, and the pose of the robot represented in this way for the task planning.

Feasibility graph. We represent the feasibility graph in ASP using the following predicates:

- `edge/3` predicate represents an edge on our graph. All edges are given as an input to the planning program. It defines there is an edge from a pose to another pose with a weight.

$$\texttt{edge}(P1, P2, W).$$

- `vertex/1` predicates are vertices on the graph.

$$\texttt{vertex}(X) \leftarrow \texttt{edge}(X, \_, \_).$$
$$\texttt{vertex}(Y) \leftarrow \texttt{edge}(\_, Y, \_).$$

Fluents and actions. To represent a dynamic system in ASP, we need to decide

for fluents to represent states, and actions to define transitions between states. To solve HCFP, we introduce one fluent and one action as follows:

- `pose/`n predicate represents the pose of the robot according to the set σ' whose cardinality equals to the total number of legs of free climbing robot.

$$\texttt{pose}(\sigma') \quad \text{and} \quad card(\sigma') = n$$

- `move/1` predicate defines the action of moving to another `pose/`n.

$$\texttt{move}(\texttt{pose}(\sigma')).$$

Initial state and goals. To represent HCFP as a planning problem, we introduce the initial and the goal states with `init/1` and `goal/1` predicates, respectively. They take a `pose/`n predicate to indicate which pose is the initial or the goal.

Domain description. Next, we represent the domain: the preconditions and effects of actions.

1. At every time step t, the free climbing robot can choose to move to at most adjacent pose $P2$ from its current pose $P1$.

$$\{\texttt{occurs}(\texttt{move}(P2), t) : \texttt{edge}(P1, P2, W)\} \leq 1 \leftarrow \texttt{holds}(P1, t), \texttt{vertex}(P1).$$

2. When the free climbing robot moves to an adjacent pose $P2$ from its current pose $P1$ at time step t, as a direct effect of this action, the pose of the robot will change to $P2$ at the next time step.

$$\texttt{holds}(P2, t+1) \leftarrow \texttt{holds}(P1, t), \texttt{occurs}(\texttt{move}(P2), t), \texttt{edge}(P1, P2, W).$$

3. Such a move action is not always possible. So the preconditions of the action should be defined.

The robot cannot move to a pose `P` if it is already at that pose.

$$\leftarrow \texttt{occurs}(\texttt{move}(P),t),\texttt{holds}(P,t).$$

The robot cannot move to a pose if it is not adjacent to its current pose.

$$\leftarrow \texttt{not edge}(P1,P2,\_),\texttt{holds}(P1,t),\texttt{occurs}(\texttt{move}(P2),t).$$

4. At every time step t, the robot should be exactly at one pose. The existence and the uniqueness is described by the following state constraint:

$$\leftarrow \#\texttt{count}\{P : \texttt{holds}(P,t)\} \neq 1.$$

5. We also need to consider what does not change over time. We define the common sense law of inertia (the pose of the robot does not change unless it is changed) as follows:

$$\{\texttt{holds}(P,t+1)\} \leftarrow \texttt{holds}(P,t).$$

6. We have to make sure that the plan reaches a goal pose C, by the following constraint:

$$\leftarrow \texttt{query}(t),\texttt{goal}(C),\texttt{not holds}(C,t).$$

7. We can further optimize the cost of the final plan as follows:

$$\#\texttt{minimize}\{W : \texttt{edge}(P1,P2,W),\texttt{holds}(P1,t),\texttt{holds}(P2,t+1)\}.$$

Multi-shot computation. For multi-shot computation, the ASP program described above should be presented to Clingo in three parts: `base`, `step`, `check`. Note that Listing 7 in Gebser et al. (2019) should be included at the beginning of the formulation.

The `base` program is grounded only once. It consists of the feasibility graph de-

scription, and the initial and goal state descriptions.

The `step` program is grounded incrementally for `t`=1,2,3,.... It consists of the domain description above (except for the last two rules).

The `check` program is grounded for each value of `t` until a solution is computed. The last two rules of the domain description above (to ensure that a goal is reached, and the optimization statement) are included in this part.

4.4 Constrained Motion Planning

With the ASP-based task planning described above, we can compute a sequence of moves of the robot from the initial state to a goal state. This hybrid plan includes information about which leg is moved to which hold at what time step, considering stability and feasibility checks. However, the feasibility of such a plan is still subject to further low-level geometric and dynamics feasibility analysis, such as, the existence of collision-free paths for all bodies of the robot, considering robot kinematics/dynamics and environmental factors, such as obstacles. Geometrically feasible of robot motions can be computed using sampling-based motion planning. Therefore, by synergistically integrating motion planning in our hybrid planning framework, we aim to generate a feasible trajectory (if exists) that the robot can actually follow.

Since the free climbing robot has similar kinematic structure with a planar parallel mechanism, and these mechanisms can be represented by mathematical equations of geometric constraints, we transform our problem into a constrained motion planning problem that can be solved by a sampling-based motion planner.

The main assumption for the motion of the free climbing robot is that it moves only one of its leg during the transition from a state to another. Therefore, the constraint equations of the robot is changing according to each leg. For the equations presented below, we assume the robot to have four legs $N = 4$, because quadrupedal robots are quite common.

The robot considered has four legs, with each leg having two degrees-of-freedom (DoF). The body of the robot is assumed to be a regular hexagon such that angles for the kinematic structure can be easily derived. The state variables for this robot can be selected as

$$\mathbf{q} = [\ \theta_0\ \ \theta_1\ \ \theta_2\ \ \theta_3\ \ \theta_4\ \ \theta_5\ \ \theta_6\ \ \theta_7\ \ x_p\ \ y_p\ \ \theta_p\ ]^T \tag{4.23}$$

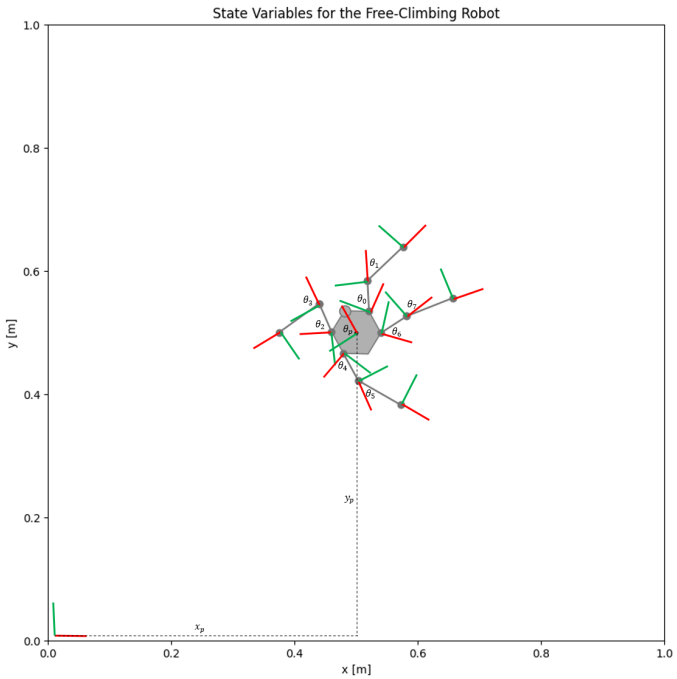


Figure 4.8 Robot Kinematics

where θ_i is the joint angle of legs, x_p and y_p are the horizontal and vertical positions for the center of the body of the robot, and θ_p represents the orientation of the body of the robot.

There are five cases considered for the motion planning:

- The first case is used when the body of the robot is required to move, while all four legs are contacting to holds. This case is beneficial when we need to change the center of the mass of the robot before moving to the other state such that the robot ensures its stability. It is also useful for inspection robots to search around a state. The constraint equations and their Jacobian matrix with respect to $\mathbf{q}$ are presented in Appedix B as Case 1.

- The second case is for the planning situation when the right front leg of the robot is allowed to move while others are attached to their respective holds. Since the right front leg is considered as the first leg in a state σ, first indices of the states in a planned sequence should be changed. The constraint equations and their Jacobian matrix with respect to $\mathbf{q}$ are presented in Appedix B as Case 2.

  Similar to the second case,

- The third case is for the left front leg, that is the second leg, so second indices of the states in a planned sequence should be changed. The related equations are in Appedix B as Case 3.
- The fourth case is for the left rear leg, that is the third leg, so third indices of the states in a planned sequence should be changed. The related equations are in Appedix B as Case 4.
- The fifth case is for the right rear leg, that is the fourth leg, so fourth indices of the states in a planned sequence should be changed. The related equations are in Appedix B as Case 5.

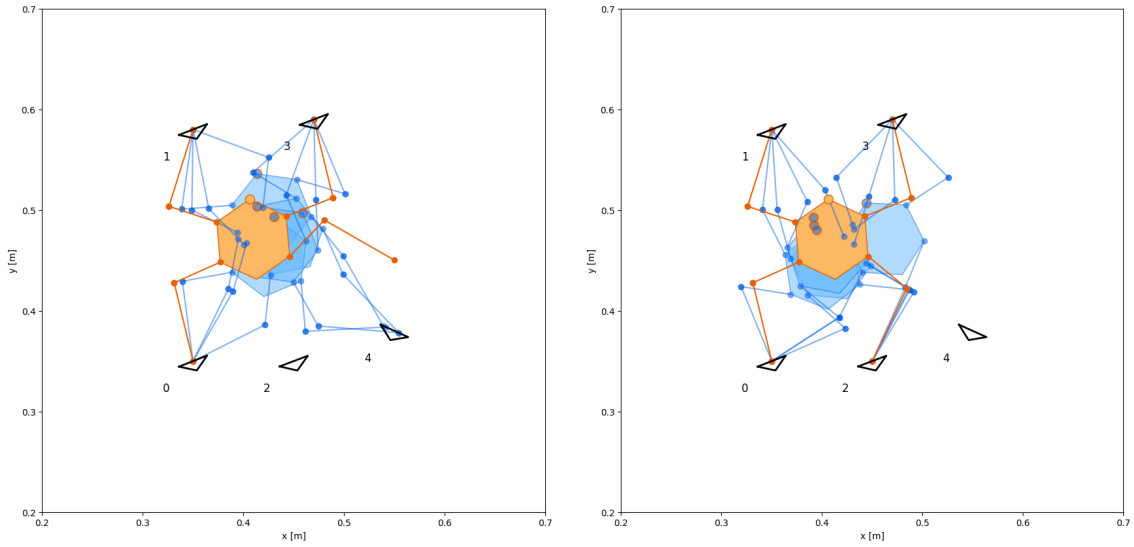
4.4.1 Goal Definition

The free climbing problem contains holds on a wall, and the robot makes its progress through supporting its legs on these holds. However, the exact location on a hold of the active leg is not decided by Task Planning, which only determines the hold. Therefore, a decision for a goal configuration has to be made by the planner. Due to the redundant degrees-of-freedom of the robot on the plane (i.e. the robot presented in Fig. 4.8 has 11 dofs, and 6 independent constraints while it is moving), there exist infinitely many configurations for this system to be placed it on a goal state.

The definition of the goal region is implemented in a different thread besides to the main planning thread, so it allows the motion planner do sample the configuration space in parallel to the goal sampling. To decide for feasible goal configuration for the body, we select a random configuration within the defined boundary, project this configuration using the Least Square Method such that our constraint equations are satisfied. Similarly, for the leg planning, the same procedure is applied, but we add two more constraints such that the active leg is on the target hold.

In Fig. 4.9, there are a number of different goal configurations are presented for both leg planning and body planning. The orange robot model shows the starting

configuration where blue models indicate possible configurations in goal regions. In Fig. 4.9a, the right rear leg is moving to the hold 4, and four different configurations are shown. Note that, the end point of the leg can be anywhere on the target hold. Similarly, in Fig. 4.9b, different configurations are presented for body while all four legs preserved their locations. In both cases, our planner enables elbow configuration changes.



(a) Possible goal configurations in a leg movement.

(b) Possible goal configurations for a body.

Figure 4.9 Sample cases for goal region.

4.5 Inverse Dynamics Control

We use an inverse dynamics controller to verify the feasibility of actuator torques required to follow the computed trajectories by computed hybrid task and motion planning.

4.5.1 Dynamic Equations

The dynamic equations of motion of the climbing robot with changing environmental contacts can be expressed as follows:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{S}^T\tau + \mathbf{J}_c^T(\mathbf{q})\lambda \tag{4.24}$$

where

- $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{j+3 \times j+3}$: the floating base matrix
- $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{j+3}$: the floating base centripetal, Corolis, and gravity forces
- $\mathbf{S} = \begin{bmatrix} \mathbf{I}_{j \times j} & | & \mathbf{0}_{j \times 3} \end{bmatrix}$: the actuated joint selection matrix
- $\tau \in \mathbb{R}^j$: the vector of actuated joint torques
- $\mathbf{J}_c \in \mathbb{R}^{l \times j+3}$: the Jacobian of l linearly independent constraints
- $\lambda \in \mathbb{R}^l$: the vector of l linearly independent constraint forces

The constraints capture the locations on the robot in contact with the environment, where external forces or torques are applied, and no motion is observed with respect to the inertial frame. Calling $\mathbf{x}_c$ the positions and orientations of these locations,

we have

$$\dot{\mathbf{x}}_c = \mathbf{J}_c\dot{\mathbf{q}} = \mathbf{0} \tag{4.25}$$

$$\ddot{\mathbf{x}}_c = \mathbf{J}_c\ddot{\mathbf{q}} + \dot{\mathbf{J}}_c\dot{\mathbf{q}} = \mathbf{0} \tag{4.26}$$

The structure of $\mathbf{J}_c$ and λ depends on the kinematic structure used to model the robot. In our case, since the environment is planar and the robot interacts with the environment through point contacts, we have 2 linearly independent constraints per leg. For 4 legs, there are 8 linearly independent constraints, $l = 8$. Also, note that there are 2 joints on each leg, so the number of joints is 8, $j = 8$. The state variables, $\mathbf{q}$, for the robot as defined in Eqn. (4.23).

4.5.2 Inverse Dynamics via Orthogonal Decomposition

Given some desired joint space motion, we wish for our robot to follow $(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)$, and given the model of robot dynamics as in Eqn. (4.24), we would like to compute actuator torques τ that can realize the desired motion, while Eqns. (4.25) and (4.26) are satisfied. As explained previously, solving for τ using Eqn. (4.24) requires full knowledge of the constraint forces, λ. However, as presented by Mistry, Buchli & Schaal (2010), when the system moves in the null space of the constraints, the total dimensionality reduces by l. Therefore, when the dynamics is represented in the reduced dimensional space, l equations can be eliminated from the full rigid-body dynamics.

The QR decomposition. Assuming $\mathbf{J}_c$ is full row rank, $\text{rank}(\mathbf{J}_c) = l$, we can compute the QR decomposition of $\mathbf{J}_c^T$ as

$$\mathbf{J}_c^T = \mathbf{Q}\begin{bmatrix}\mathbf{R}\\ \mathbf{0}\end{bmatrix} \tag{4.27}$$

Multiplying Eqn. (4.24) by $\mathbf{Q}^T$ decomposes the rigid-body dynamics into two inde-

pendent equation as

$$\mathbf{S}_c\mathbf{Q}^T(\mathbf{M}\ddot{\mathbf{q}}+\mathbf{h}) = \mathbf{S}_c\mathbf{Q}^T\mathbf{S}^T\tau + \mathbf{R}\lambda \tag{4.28}$$

$$\mathbf{S}_u\mathbf{Q}^T(\mathbf{M}\ddot{\mathbf{q}}+\mathbf{h}) = \mathbf{S}_u\mathbf{Q}^T\mathbf{S}^T\tau \tag{4.29}$$

where

$$\mathbf{S}_c = \begin{bmatrix} \mathbf{I}_{l\times l} & | & \mathbf{0}_{l\times(j+3-l)} \end{bmatrix} \tag{4.30}$$

$$\mathbf{S}_u = \begin{bmatrix} \mathbf{0}_{(j+3-l)\times l} & | & \mathbf{I}_{(j+3-l)\times(j+3-l)} \end{bmatrix} \tag{4.31}$$

Control Equation. Now, the one can compute the control torques using Eqn. (4.29) and a pseudo-inverse as

$$\tau = (\mathbf{S}_u\mathbf{Q}^T\mathbf{S}^T)^+\mathbf{S}_u\mathbf{Q}^T(\mathbf{M}\ddot{\mathbf{q}}+\mathbf{h}) \tag{4.32}$$

Calculating Contact Forces. Also, the contact forces can be calculated according to Eqn. (4.28):

$$\lambda = \mathbf{R}^{-1}\mathbf{S}_c\mathbf{Q}^T(\mathbf{M}\ddot{\mathbf{q}}+\mathbf{h}-\mathbf{S}^T\tau) \tag{4.33}$$

Joint Tracking with Feed-forward Compensation. PD control based joint space tracking with feed-forward compensation can be implemented by a controller of the form:

$$\tau = \mathrm{InvDyn}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_d) + \mathbf{K}_P\mathbf{S}(\mathbf{q}_d-\mathbf{q}) + \mathbf{K}_D\mathbf{S}(\dot{\mathbf{q}}_d-\dot{\mathbf{q}}) \tag{4.34}$$

where $\mathbf{K}_P$ and $\mathbf{K}_D$ are position and velocity gain matrices and $\mathrm{InvDyn}(\cdot)$ is calculated according to Eqn. (4.32).

5. SIMULATION-BASED EXPERIMENTAL EVALUATIONS

The methods described in Chapter 4 are evaluated on several challenging benchmark scenarios. We evaluate our method's performance in terms of scalability by increasing climbing hold numbers in problem instances.

All simulations were performed on workstation with an Intel Xeon Gold 2 x 6130 CPU running at 3.70 GHz using a single thread and 32 GB RAM. All algorithms were implemented in PYTHON, with SCIPY tools (Virtanen et al., 2020). A timeout of 600 seconds per trial was imposed for each instance of motion planning.

5.1 Sample Scenarios

To demonstrate the capability of the Hybrid Planning to solve a large variety of problems, we have also illustrated its applications with several difficult benchmark scenarios. For each scenario, the robot model presented in Chapter 4 is used, and a climbing environment is designed with various number of holds, whose slope angles and coefficients of static friction are distributed through normal distribution of random samples.

5.1.1 Scenario 1: Basic Free Climbing

The first scenario is a basic free climbing wall designed to demonstrate that the proposed method works. As shown in Figure 5.1, there are 10 holds distributed in a horizontal direction. Each hold has the same physical shape and dimension, but the coefficients of static friction of the holds and their slopes are chosen randomly from a feasible range. Therefore, angles of friction cones vary based on the friction coefficient.

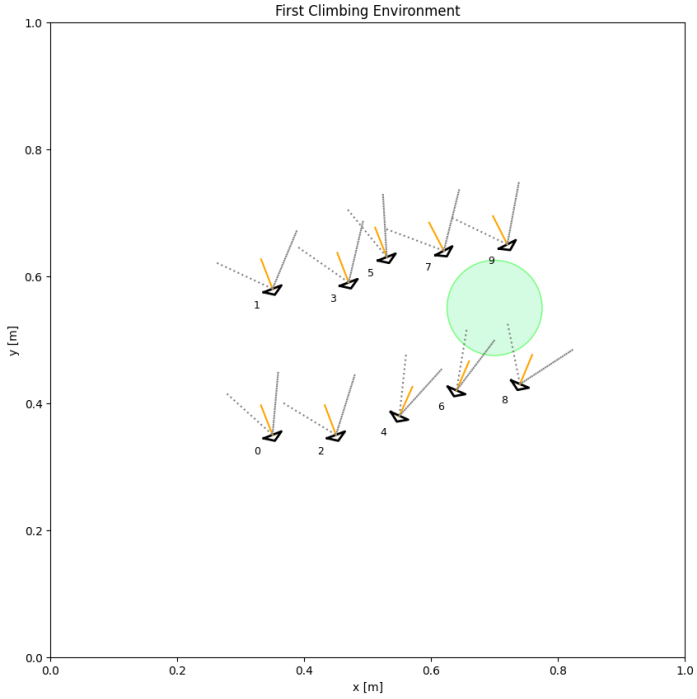


Figure 5.1 Basic Free Climbing Environment

The robot is requested to move from left side of the wall to the right side of the wall. In other words, the initial state is $\sigma_0 = (3, 1, 0, 2)$, and the final position for the pelvis of the robot is $(x_p^f, y_p^f) = (0.7, 0.55)$ m. The first step is generating all possible states using the State Creator as described in Section 4.2.1. Next, we eliminate the ones that do not satisfy the feasibility checks related to the states. Using Eqn. (4.1), there are 5040 states to be checked.

$$\mathcal{H} = \{0, 1, ..., 9\}$$
$$n(\Sigma') = {}^{\mathcal{H}}P_n = {}^{10}P_4 = \frac{10!}{(10-4)!} = 5040$$

The number of possible states are first reduced to 720 by the State Validator. Then, the number of possible states decreases to 104 after the Leg Crossing Check. That is, there exist 104 feasible states for the basic climbing environment that the robot can be placed considering its kinematics.

Next, we construct the graph as presented in Section 4.2.2. Starting from the initial state $\sigma_i = (3, 1, 0, 2)$ as the first node of the graph, we check if adjacent states are one of the 104 feasible states, and whether there exists a feasible transition between these states. For this scenario, there are 85 feasible adjacent states, and 37 of them are unique states. After the Reachability Check and the Stability Check, 29 feasible transitions are found. As a result, the graph of this scenario has 29 edges with 24 nodes, and can be visualized as in Figure 5.2.

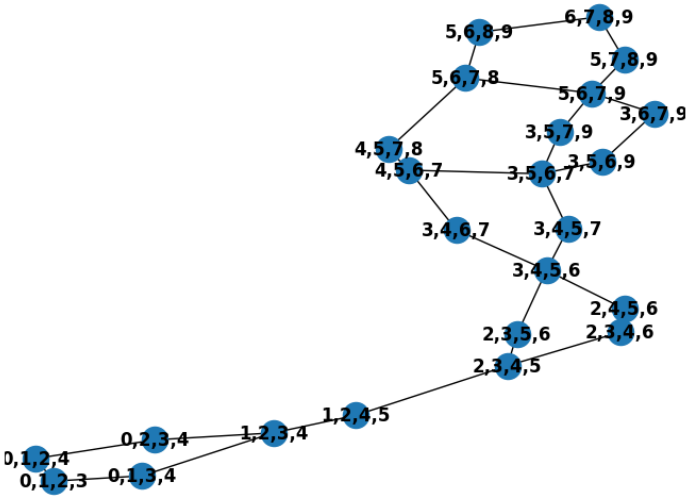


Figure 5.2 First Climbing Environment converted graph

Next, the final position for the body of the robot, which is desired to be at $(x_p^f, y_p^f) = (0.7, 0.55)$ m is determined to correspond to the region enclosed by the holds $(6, 7, 8, 9)$. Therefore, this pose is provided as the goal for the task planner.

The multi-shot ASP Task Planning computes a solution with a minimum makespan

of 12 time steps in 0.024 seconds. A plan, with its history, is presented in the following form

```
holds(pose(0,1,2,3),0) holds(pose(0,1,3,4),1)
occurs(move(pose(0,1,3,4)),0) holds(pose(1,2,3,4),2)
occurs(move(pose(1,2,3,4)),1) holds(pose(1,2,4,5),3)
occurs(move(pose(1,2,4,5)),2) holds(pose(2,3,4,5),4)
occurs(move(pose(2,3,4,5)),3) holds(pose(2,3,5,6),5)
occurs(move(pose(2,3,5,6)),4) holds(pose(3,4,5,6),6)
occurs(move(pose(3,4,5,6)),5) holds(pose(3,4,6,7),7)
occurs(move(pose(3,4,6,7)),6) holds(pose(4,5,6,7),8)
occurs(move(pose(4,5,6,7)),7) holds(pose(4,5,7,8),9)
occurs(move(pose(4,5,7,8)),8) holds(pose(5,6,7,8),10)
occurs(move(pose(5,6,7,8)),9) holds(pose(5,6,8,9),11)
occurs(move(pose(5,6,8,9)),10) holds(pose(6,7,8,9),12)
occurs(move(pose(6,7,8,9)),11)
Optimization: 753
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 753
Calls        : 13
Time         : 0.024s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.024s
```

Since we know the initial state as $\sigma_0 = (3,1,0,2)$, we can decode the output of ASP. For example, at step 1, the robot is on `pose(0,1,3,4)`. Since the robot is allowed to move only one leg from pose to pose, we can find out that the different hold number is 4, and the next state is $\sigma_1 = (3,1,0,4)$. Then the sequence of states that the robot follows by the plan above can be extracted as follows:

$$\Sigma = \Big\{(3,1,0,2),(3,1,0,4),(3,1,2,4),(5,1,2,4),(5,3,2,4),(5,3,2,6),(5,3,4,6),\\ (7,3,4,6),(7,5,4,6),(7,5,4,8),(7,5,6,8),(9,5,6,8),(9,7,6,8)\Big\}$$

At this point, motion planning needs to be employed to check whether the task plan is feasible, or not. As described in Section 4.4, the robot is modelled as a parallel manipulator. Therefore, we can define geometric constraints for our system and use Constrained Sampling-based Planners to solve for a motion plan from state by state.

Figure 5.3 presents snapshots from the animation of the robot free climbing on the

environment. As described in the problem, the robot is desired to move from left side of the environment to the right side. Shaded regions in each figure represents the support polygon where the center of mass of the robot should lie to ensure stability. RRTConnect is used as the motion planner as it is observed to perform better in terms of time efficiency.

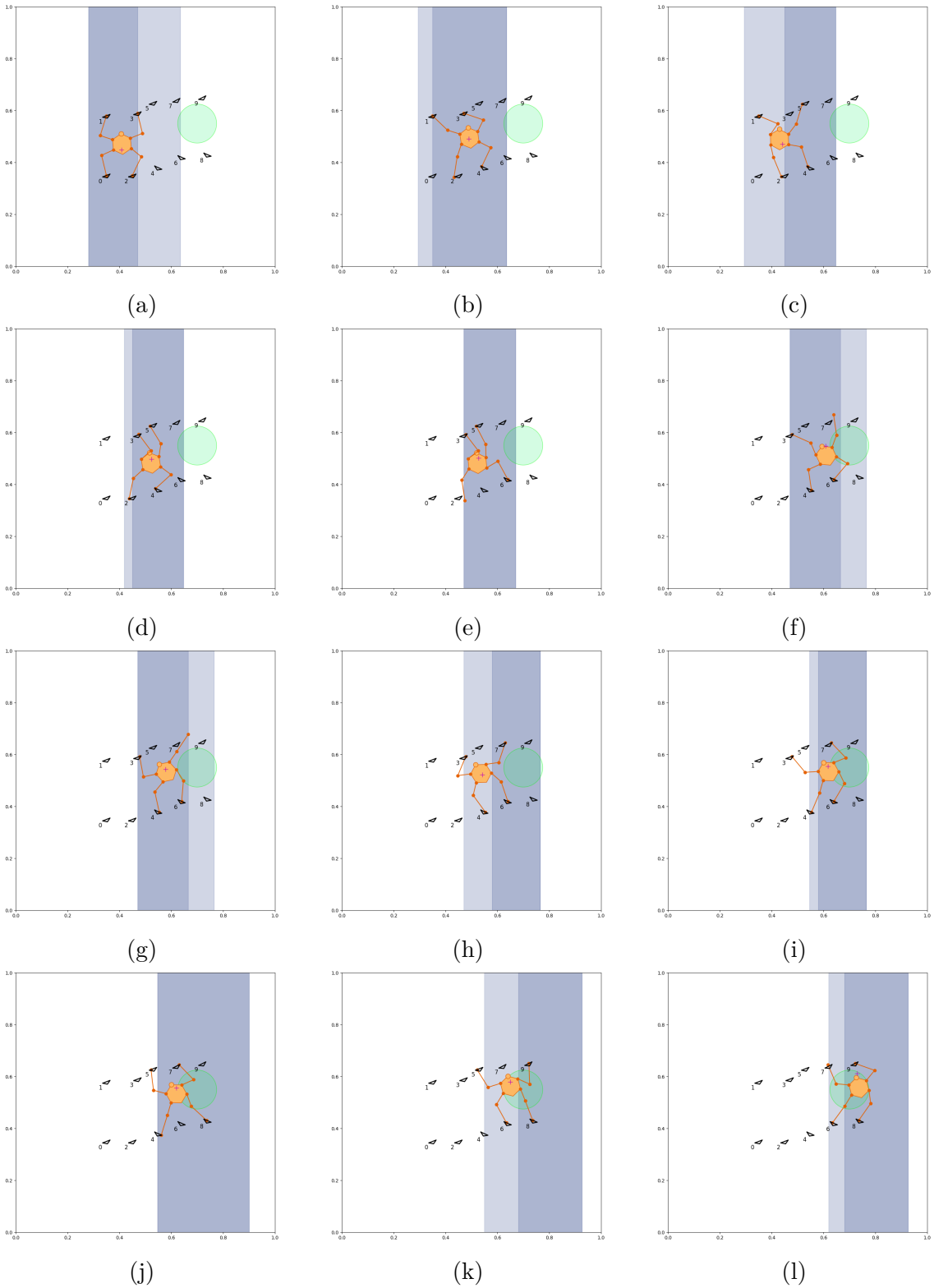
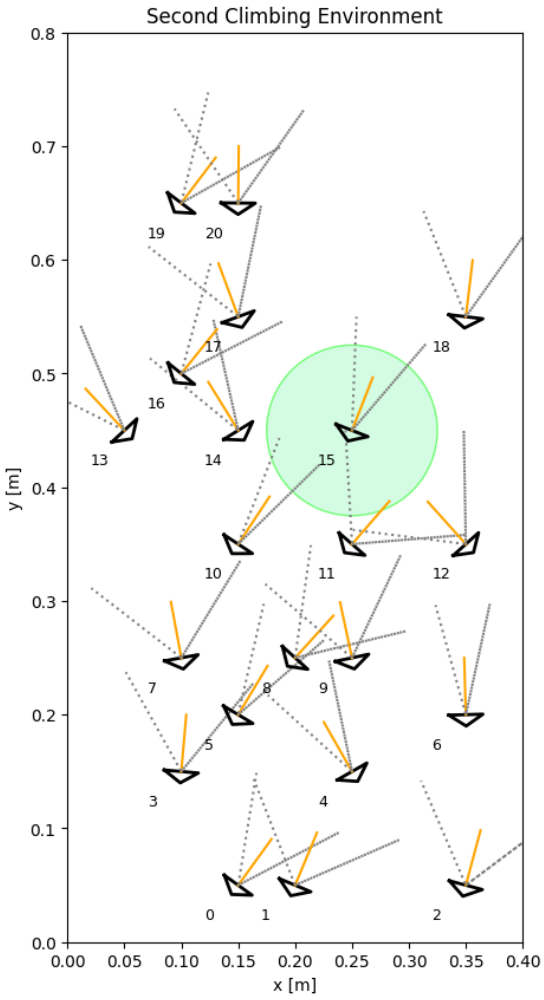


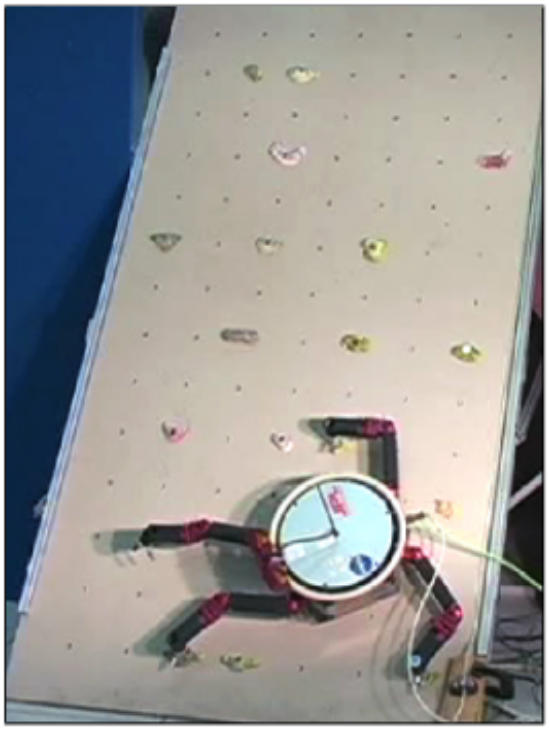
Figure 5.3 Snapshots for the motion plan of the first scenario

5.1.2 Scenario 2: A Higher Free Climb

The second scenario involves a more complex climbing wall inspired by the climbing wall used by Bretl (2005). As shown in Figure 5.4b, the climbing environment is created by distribution of climbing holds in an environment. As we can estimate from the figure, the distance between each hold is close to the length of the front link of the robot. Then, we place the holds with the same distances based on our robot. Similarly in all environments, the slope and the coefficient of static frictions are chosen randomly as presented in Figure 5.4a.



(a) Our adaptation



(b) Base map adapted from Bretl (2005)

Figure 5.4 Second Climbing Environment

The robot is assigned to move from lower side of the climbing wall to the top of the wall. The initial state is $\sigma_i = (9, 3, 0, 2)$, and the final position for the pelvis of the

robot, $(x_p^f, y_p^f) = (0.25, 0.45)$ m. First step is to generate all possible states using State Creator as described in Section 4.2.1. After that, we eliminate the ones that do not satisfy the feasibility checks related to the states.

Using Eqn. (4.1), there are 143640 states to be checked.

$$\mathcal{H} = \{0, 1, ..., 20\}$$
$$n(\Sigma') = {}^{\mathcal{H}}P_n = {}^{21}P_4 = \frac{21!}{(21-4)!} = 143640$$

The number of possible states are reduced to 17352 by the State Validator. Then, it is decreased to 1842 after Leg Crossing Check. That is, there are 1842 feasible states for this climbing environment that the robot can be placed considering its kinematics.

Next, we construct the graph as presented in Section 4.2.2. Starting from the initial state $\sigma_i = (9, 3, 0, 2)$ as the first node to the graph, we check if adjacent states are one of the 1842 feasible states, and whether there is a feasible transition between states. For this scenario, there are 18150 feasible adjacent states, and 2368 of them are unique states. After the Reachability Check, this number is reduced to 2289. Then, the Stability Check eliminates them to 1937 edges. As a result, the graph has 1937 edges with 442 nodes.

Then, it is determined that the final position for the body of the robot, which is desired to be at $(x_p^f, y_p^f) = (0.25, 0.45)$ m, corresponds to the region enclosed by $(10, 15, 17, 18)$ holds. Therefore, we this pose is assigned as the goal to the task planner.

The multi-shot ASP Task Planning computes a solution wit minimum makespan of 9 time steps in around 1 second. A plan, with its history, is presented in the following form

```
holds(pose(0,2,3,9),0)  holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0)  holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1)  holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2)  holds(pose(5,6,7,15),4)
occurs(move(pose(5,6,7,15)),3)  holds(pose(5,7,12,15),5)
```

```
occurs(move(pose(5,7,12,15)),4) holds(pose(5,12,14,15),6)
occurs(move(pose(5,12,14,15)),5) holds(pose(10,12,14,15),7)
occurs(move(pose(10,12,14,15)),6) holds(pose(10,12,15,17),8)
occurs(move(pose(10,12,15,17)),7) holds(pose(10,12,17,18),9)
occurs(move(pose(10,12,17,18)),8) holds(pose(10,15,17,18),10)
occurs(move(pose(10,15,17,18)),9)
Optimization: 630
OPTIMUM FOUND

Models       : 4
  Optimum    : yes
Optimization : 630
Calls        : 11
Time         : 1.246s (Solving: 0.01s 1st Model: 0.01s Unsat: 0.00s)
CPU Time     : 1.246s
```

Starting from the initial state $\sigma_0 = (9,3,0,2)$, we can decode the output of the ASP solution. Then the sequence of states that the robot follows by the plan above can be extracted as follows:

$$\Sigma = \Big\{(9,3,0,2),(9,3,0,6),(9,7,0,6),(9,7,5,6),$$
$$(15,7,5,6),(15,7,5,12),(15,14,5,12),(15,14,10,12),$$
$$(15,17,10,12),(18,17,10,12),(18,17,10,15)\Big\}$$

Fig. 5.5 presents snapshots from the animation of the robot. As described in the problem, the robot is desired to move from lower part of the environment to the top. Shaded regions on each figure represents the support polygon where the center of mass of the robot should lie to ensure balance.

5.1.2.1 Not Optimum Task Plan

This problem instance is solved without any optimization in Task Planning. Therefore, the related ASP program gives four different plans, with their histories, as

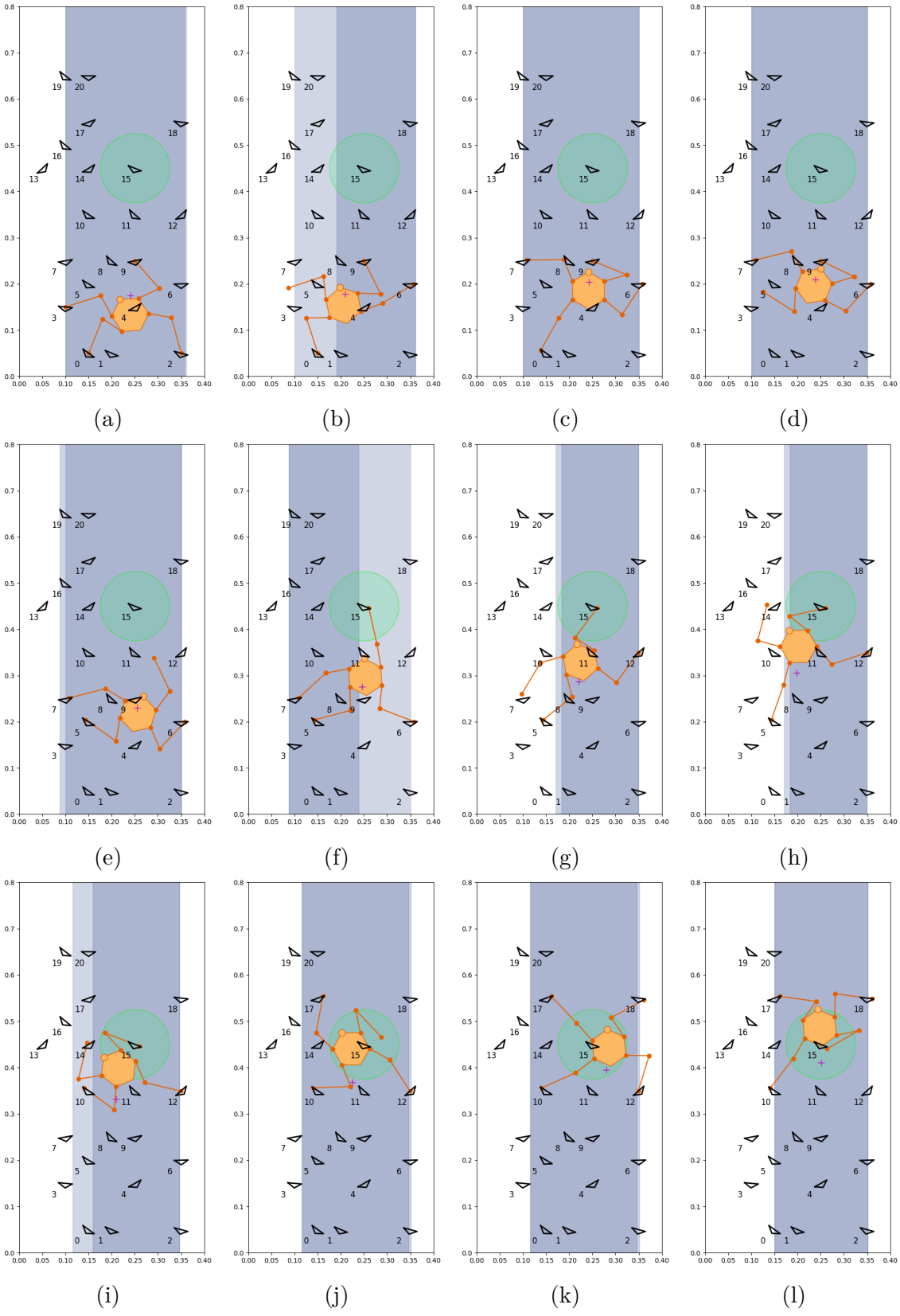


Figure 5.5 Snapshots for the motion plan of the second scenario

presented below. Note that the Answer 4 is our the optimum plan. In order to show the affect of the optimization, lets choose another plan such in Answer 1.

```
Answer: 1
holds(pose(0,2,3,9),0) holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0) holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1) holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2) holds(pose(5,6,7,15),4)
occurs(move(pose(5,6,7,15)),3) holds(pose(5,7,11,15),5)
occurs(move(pose(5,7,11,15)),4) holds(pose(5,11,13,15),6)
occurs(move(pose(5,11,13,15)),5) holds(pose(10,11,13,15),7)
occurs(move(pose(10,11,13,15)),6) holds(pose(10,11,15,17),8)
occurs(move(pose(10,11,15,17)),7) holds(pose(10,11,17,18),9)
occurs(move(pose(10,11,17,18)),8) holds(pose(10,15,17,18),10)
occurs(move(pose(10,15,17,18)),9)
Answer: 2
holds(pose(0,2,3,9),0) holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0) holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1) holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2) holds(pose(5,6,7,15),4)
occurs(move(pose(5,6,7,15)),3) holds(pose(5,7,12,15),5)
occurs(move(pose(5,7,12,15)),4) holds(pose(5,12,14,15),6)
occurs(move(pose(5,12,14,15)),5) holds(pose(10,12,14,15),7)
occurs(move(pose(10,12,14,15)),6) holds(pose(10,12,14,18),8)
occurs(move(pose(10,12,14,18)),7) holds(pose(10,14,15,18),9)
occurs(move(pose(10,14,15,18)),8) holds(pose(10,15,17,18),10)
occurs(move(pose(10,15,17,18)),9)
Answer: 3
holds(pose(0,2,3,9),0) holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0) holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1) holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2) holds(pose(5,6,7,15),4)
occurs(move(pose(5,6,7,15)),3) holds(pose(5,7,12,15),5)
occurs(move(pose(5,7,12,15)),4) holds(pose(5,12,14,15),6)
occurs(move(pose(5,12,14,15)),5) holds(pose(10,12,14,15),7)
occurs(move(pose(10,12,14,15)),6) holds(pose(10,12,14,18),8)
occurs(move(pose(10,12,14,18)),7) holds(pose(10,12,17,18),9)
occurs(move(pose(10,12,17,18)),8) holds(pose(10,15,17,18),10)
occurs(move(pose(10,15,17,18)),9)
Answer: 4
holds(pose(0,2,3,9),0) holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0) holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1) holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2) holds(pose(5,6,7,15),4)
occurs(move(pose(5,6,7,15)),3) holds(pose(5,7,12,15),5)
occurs(move(pose(5,7,12,15)),4) holds(pose(5,12,14,15),6)
occurs(move(pose(5,12,14,15)),5) holds(pose(10,12,14,15),7)
occurs(move(pose(10,12,14,15)),6) holds(pose(10,12,15,17),8)
occurs(move(pose(10,12,15,17)),7) holds(pose(10,12,17,18),9)
```

```
occurs(move(pose(10,12,17,18)),8) holds(pose(10,15,17,18),10)
occurs(move(pose(10,15,17,18)),9)
SATISFIABLE

Models       : 4
Calls        : 11
Time         : 1.371s (Solving: 0.01s 1st Model: 0.01s Unsat: 0.00s)
CPU Time     : 1.371s
```

Then the sequence of states that the robot follows by the plan for Answer 1 can be extracted as follows

$$\Sigma = \Big\{(9,3,0,2),(9,3,0,6),(9,7,0,6),(9,7,5,6),$$
$$(15,7,5,6),(15,7,5,11),(15,14,5,11),(15,14,10,11),$$
$$(18,14,10,11),(18,17,10,11),(18,17,10,15)\Big\}$$

The snapshots for the not optimal task plan are presented in Fig 5.6. As one can see, the robot cannot go to the next state after the state $(15,7,5,11)$ because when the robot releases its left front leg, the stable region will be a darker shaded area in Fig. 5.6l. Since this area is too narrow, the robot should not move in this configuration. Hence, the task planner should avoid this edge on the graph. If the hybrid planner encounters such a case, and the motion planner cannot find a motion plan, it calls the replanning loop that cuts the edge on the feasibility graph and gives guidance to the task planner.

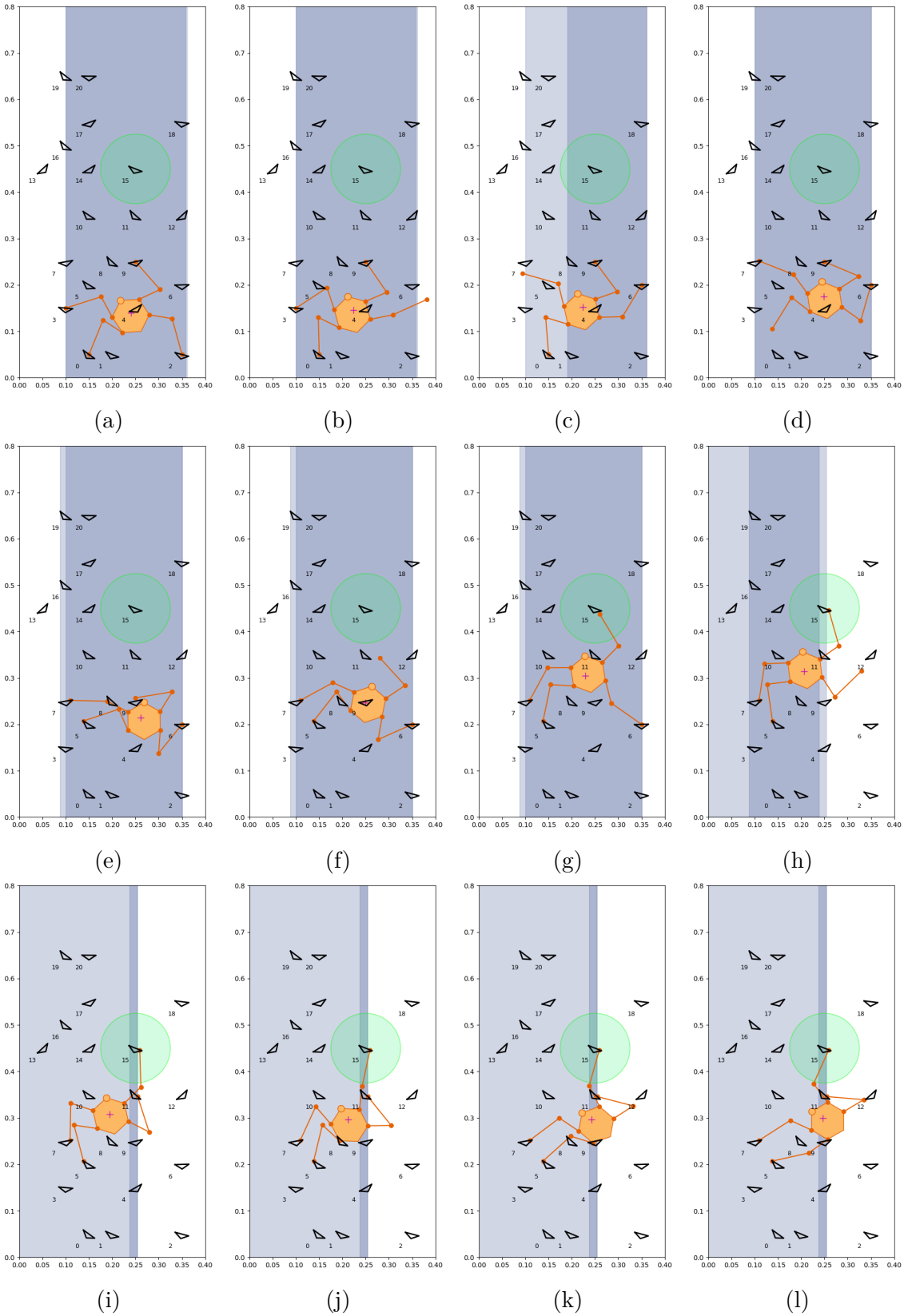


Figure 5.6 Snapshots for the not optimum task plan

5.1.3 Scenario 3: A Higher and Wider Free Climb

The third scenario is an expanded version of the second scenario, where the environment is wider. As shown in Figure 5.7, the hold of previous environment are mirrored to study the scalability of our method.

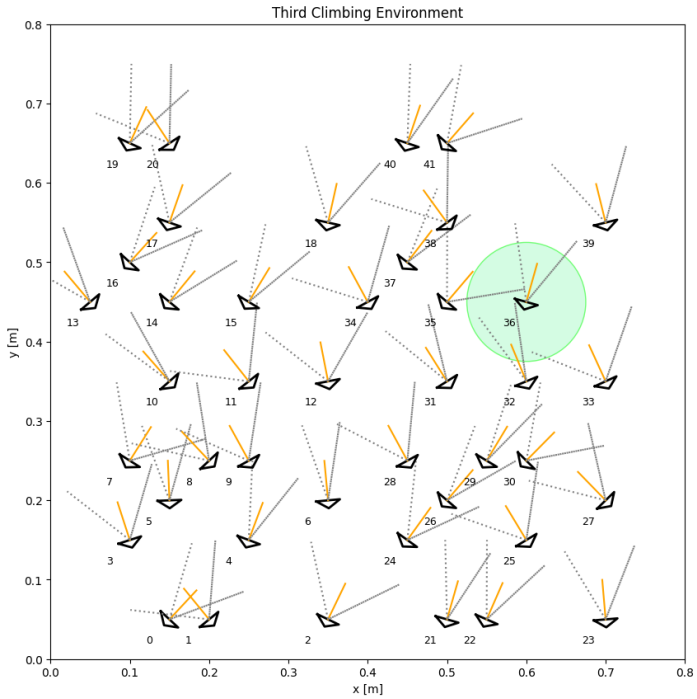


Figure 5.7 Third Climbing Environment

The robot is assigned to move from lower left side of the wall to the upper right side of the wall. In other words, it moves through the diagonal. The initial state is $\sigma_0 = (9,3,0,2)$, and the final position for the pelvis of the robot, $(x_p^f, y_p^f) = (0.60, 0.45)$ m.

Using Eqn. (4.1), there are 2686320 states to be checked.

$$\mathcal{H} = \{0,1,...,41\}$$
$$n(\Sigma') = {}^{\mathcal{H}}P_n = {}^{42}P_4 = \frac{42!}{(42-4)!} = 2686320$$

The number of feasible states reduces after each feasibility check as follows:

- State Validator: 54096,

- Leg Crossing: 5812.

For the graph construction,

- 63187 feasible transitions exist,
- 7804 of them are unique,
- 7352 of them satisfy the Reachability Check, and
- 6396 of them satisfy the Stability Check.

As a result, the graph has 6396 edges with 1387 nodes.

The final position for the body of the robot, which is desired to reach $(x_p^f, y_p^f) = (0.60, 0.45)$ m, corresponds to the region enclosed by $(33, 37, 38, 39)$ holds. Therefore, we can give this pose as the goal to the task planner.

The multi-shot ASP Task Planning computes a solution with minimum makespan of 15 time steps in about 24 seconds. A plan, with its history, is presented in the following form

```
holds(pose(0,2,3,9),0) holds(pose(0,3,6,9),1)
occurs(move(pose(0,3,6,9)),0) holds(pose(0,6,7,9),2)
occurs(move(pose(0,6,7,9)),1) holds(pose(5,6,7,9),3)
occurs(move(pose(5,6,7,9)),2) holds(pose(5,6,7,12),4)
occurs(move(pose(5,6,7,12)),3) holds(pose(5,6,12,15),5)
occurs(move(pose(5,6,12,15)),4) holds(pose(6,9,12,15),6)
occurs(move(pose(6,9,12,15)),5) holds(pose(6,9,15,34),7)
occurs(move(pose(6,9,15,34)),6) holds(pose(9,15,31,34),8)
occurs(move(pose(9,15,31,34)),7) holds(pose(12,15,31,34),9)
occurs(move(pose(12,15,31,34)),8) holds(pose(12,15,31,38),10)
occurs(move(pose(12,15,31,38)),9) holds(pose(12,31,37,38),11)
occurs(move(pose(12,31,37,38)),10) holds(pose(12,36,37,38),12)
occurs(move(pose(12,36,37,38)),11) holds(pose(32,36,37,38),13)
occurs(move(pose(32,36,37,38)),12) holds(pose(32,37,38,39),14)
occurs(move(pose(32,37,38,39)),13) holds(pose(33,37,38,39),15)
occurs(move(pose(33,37,38,39)),14)
Optimization: 627
OPTIMUM FOUND

Models       : 6
  Optimum    : yes
Optimization : 627
Calls        : 16
Time         : 22.807s (Solving: 0.53s 1st Model: 0.04s Unsat: 0.37s)
```

CPU **Time** : 22.805 s

Then the sequence of states that the robot follows by the plan above can be extracted as follows

$$\Sigma = \Big\{(9,3,0,2),(9,3,0,6),(9,7,0,6),(9,7,5,6),(12,7,5,6),(12,15,5,6),(12,15,9,6),$$
$$(34,15,9,6),(34,15,9,31),(34,15,12,31),(38,15,12,31),(38,37,12,31),(38,37,12,36),$$
$$(38,37,32,36),(38,37,32,39),(38,37,33,39)\Big\}$$

Fig. 5.8 presents snapshots from the animation of the robot. As described in the problem, the robot is moving from left lower corner to the right upper corner. Shaded regions on each figure represents the support polygon where the center of mass of the robot should lie to ensure balance.

At the initial attempt for this planner, the motion planner couldn't find a feasible continuous trajectory for the robot moving on `edge(pose(12,36,37,38), pose(35,36,37,38), 76)`. This can be related the configuration of the robot at `pose(12,36,37,38)`, or the distance to the goal in configuration space is too large such that the planner couldn't reach. Or, the motion planner couldn't find any goal configuration for `pose(35,36,37,38)`. Therefore, the replanning loop is called and it removed this edge from the feasibility graph. The plan presented in Fig 5.1.3 is the result of the second attempt.

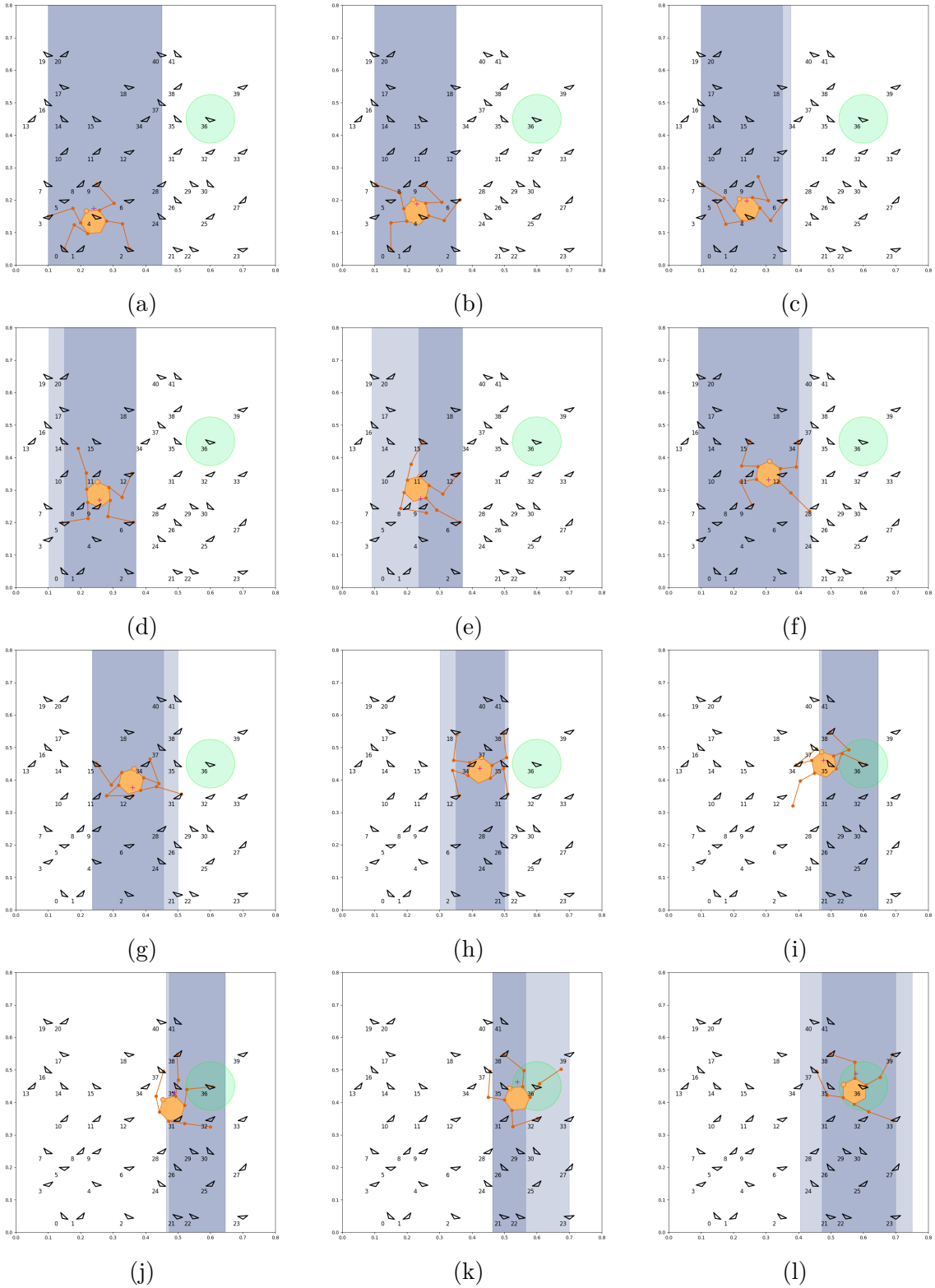


Figure 5.8 Snapshots for the motion plan of the third scenario

5.1.4 Scenario 4: Scenario with a Must-be-visited Region

This scenario is the climbing environment with the Higher and Wider Free Climb scenario, but there is a must-be-visited region (waypoint) where the robot should visit before it reaches the goal location. The main target is to show that the task planner can solve complex planning problems.

The same feasibility analysis is made as it is the same climbing environment with the Section 5.1.3, but the goal region defined at $(x_p^f, y_p^f) = (0.60, 0.45)$ m corresponding to the state $(29, 33, 36, 37)$, and the waypoint at $\mathcal{W} = \{(x_p^0, y_p^0)\} = \{(0.60, 0.15)\}$ m. The scenario is shown in Fig. 5.9. The blue and the green circles represent the waypoint and the goal, respectively.

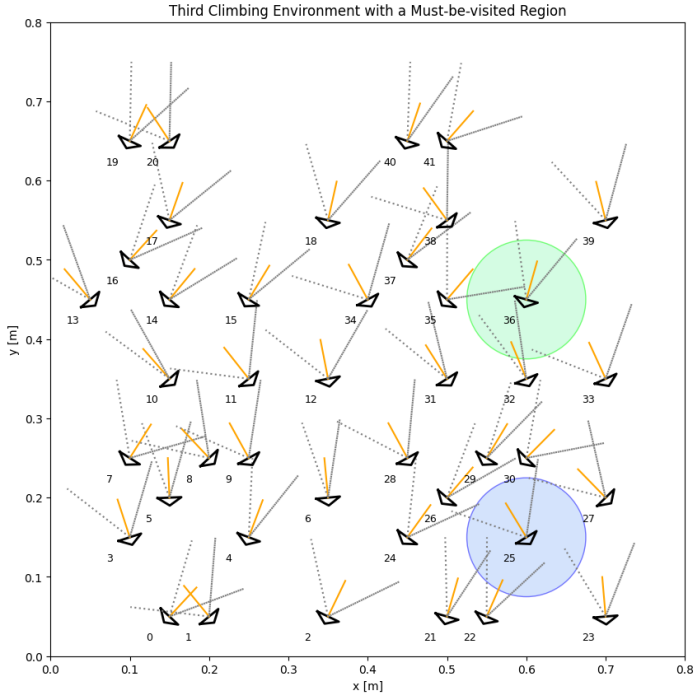


Figure 5.9 Third Climbing Environment with a Must-be-visited Region

The feasibility analysis of the environment is same as shown in Section 5.1.3 because the environment and the robot is same. However, a small modification to Task Planning is done in order to plan with the waypoint. First define the `waypoint/1` predicate representing the waypoint pose on the feasibility graph,

$$\texttt{waypoint}(P).$$

Then, make sure that the plan contains the waypoint W at anytime, by the following constraint,

$$\leftarrow \texttt{query}(t), \texttt{waypoint}(W), \texttt{not holds}(W, \_).$$

Then, the optimum plan, with its history, is presented in the following form,

```
holds(pose(0,2,3,9),0) holds(pose(0,2,8,9),1)
occurs(move(pose(0,2,8,9)),0) holds(pose(2,4,8,9),2)
occurs(move(pose(2,4,8,9)),1) holds(pose(2,4,8,24),3)
occurs(move(pose(2,4,8,24)),2) holds(pose(2,4,24,28),4)
occurs(move(pose(2,4,24,28)),3) holds(pose(2,6,24,28),5)
occurs(move(pose(2,6,24,28)),4) holds(pose(2,6,28,29),6)
occurs(move(pose(2,6,28,29)),5) holds(pose(6,25,28,29),7)
occurs(move(pose(6,25,28,29)),6) holds(pose(22,25,28,29),8)
occurs(move(pose(22,25,28,29)),7) holds(pose(22,27,28,29),9)
occurs(move(pose(22,27,28,29)),8) holds(pose(26,27,28,29),10)
occurs(move(pose(26,27,28,29)),9) holds(pose(26,27,28,33),11)
occurs(move(pose(26,27,28,33)),10) holds(pose(26,27,33,36),12)
occurs(move(pose(26,27,33,36)),11) holds(pose(27,31,33,36),13)
occurs(move(pose(27,31,33,36)),12) holds(pose(29,31,33,36),14)
occurs(move(pose(29,31,33,36)),13) holds(pose(29,33,36,37),15)
occurs(move(pose(29,33,36,37)),14)
Optimization: 786
OPTIMUM FOUND

Models       : 4
  Optimum    : yes
Optimization : 786
Calls        : 16
Time         : 50.154s (Solving: 25.96s 1st Model: 2.20s Unsat: 21.03s)
CPU Time     : 50.132s
```

Then the sequence of states that the robot follows by the plan above can be extracted as follows

$$\begin{aligned}\Sigma = \Big\{&(9,3,0,2),(9,8,0,2),(9,8,4,2),(24,8,4,2),\\ &(24,28,4,2),(24,28,6,2),(29,28,6,2),(29,28,6,25),\\ &(29,28,22,25),(29,28,22,27),(29,28,26,27),(33,28,26,27),\\ &(33,36,26,27),(33,36,31,27),(33,36,31,29),(33,36,37,29)\Big\}\end{aligned}$$

The optimum result after obtained with RRT* solution is presented in Fig. 5.10

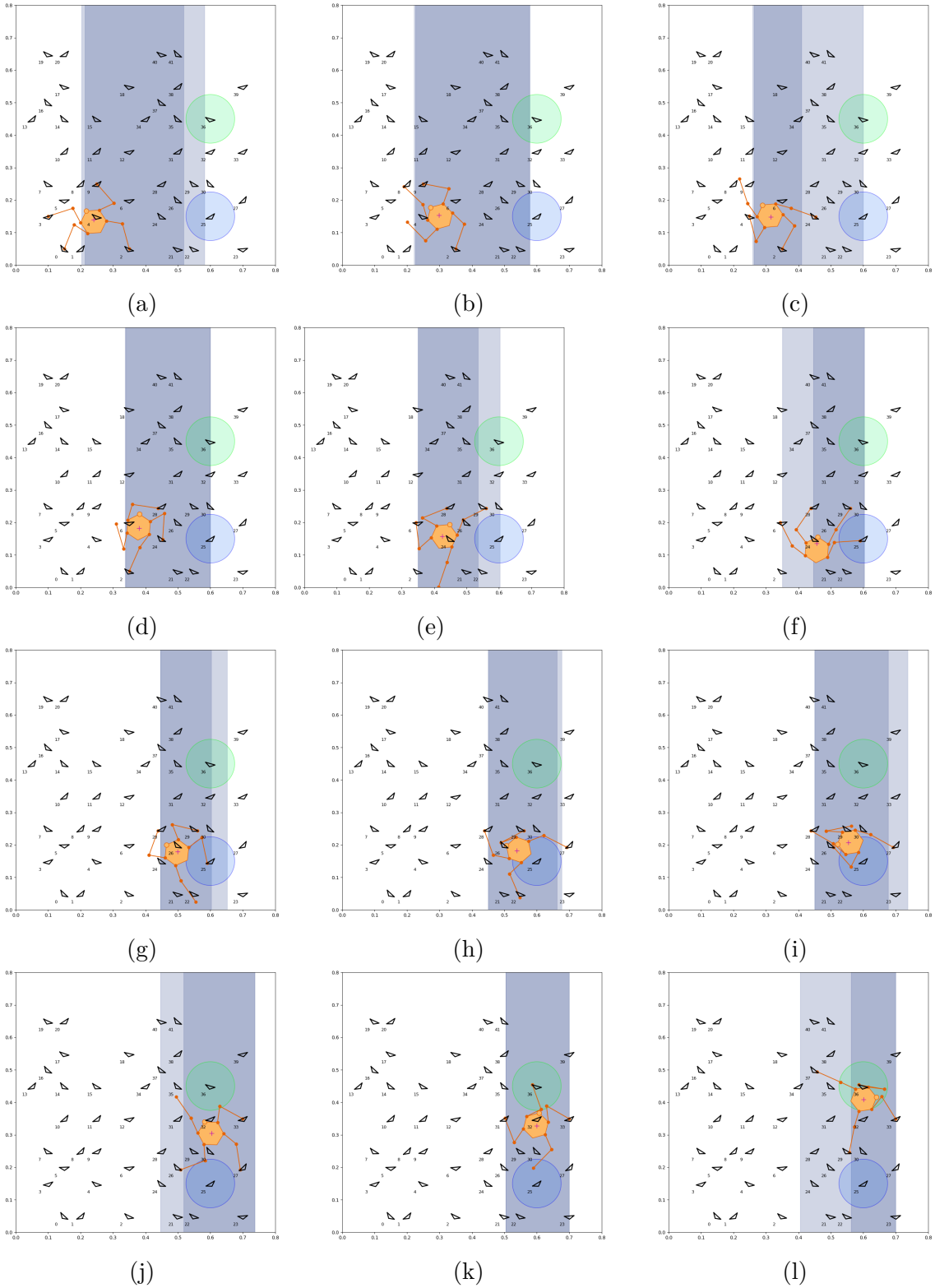


Figure 5.10 Snapshots for the task plan of the scenario with two robots

5.1.5 Scenario 5: Scenario with Two Robots

This scenario is the climbing environment with the Higher and Wider Free Climb scenario, but there are two robots starting from lower corners to traverse diagonals.

The same feasibility analysis is made as it is the same climbing environment with the Section 5.1.3, but goal regions defined at $(x_p^f, y_p^f) = (0.60, 0.45)$ m, and $(x_p^f, y_p^f) = (0.25, 0.45)$ m, respectively as presented in Fig. 5.11.

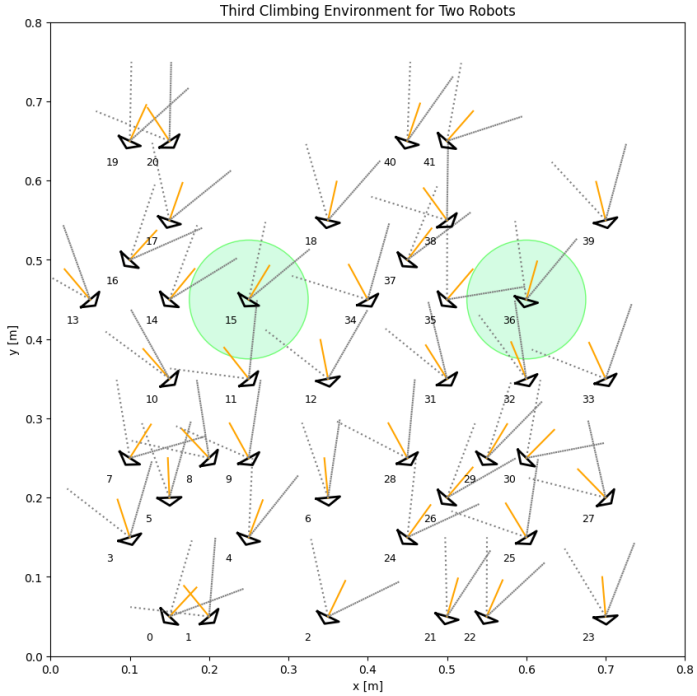


Figure 5.11 Third Climbing Environment with Two Robots

Then, the optimum plan, with its history, is presented in the following form,

```
holds(1,pose(0,2,3,9),0)  holds(2,pose(22,23,27,30),0)
holds(2,pose(23,27,29,30),1)  occurs(2,move(pose(23,27,29,30)),0)
holds(1,pose(0,3,6,9),1)  occurs(1,move(pose(0,3,6,9)),0)
holds(1,pose(0,6,7,9),2)  occurs(1,move(pose(0,6,7,9)),1)
holds(2,pose(24,27,29,30),2)  occurs(2,move(pose(24,27,29,30)),1)
holds(2,pose(24,27,28,30),3)  occurs(2,move(pose(24,27,28,30)),2)
holds(1,pose(5,6,7,9),3)  occurs(1,move(pose(5,6,7,9)),2)
holds(1,pose(5,6,7,12),4)  occurs(1,move(pose(5,6,7,12)),3)
holds(2,pose(24,27,28,31),4)  occurs(2,move(pose(24,27,28,31)),3)
holds(2,pose(24,26,28,31),5)  occurs(2,move(pose(24,26,28,31)),4)
holds(1,pose(5,6,12,15),5)  occurs(1,move(pose(5,6,12,15)),4)
holds(1,pose(6,11,12,15),6)  occurs(1,move(pose(6,11,12,15)),5)
holds(2,pose(9,24,26,31),6)  occurs(2,move(pose(9,24,26,31)),5)
holds(2,pose(9,12,24,26),7)  occurs(2,move(pose(9,12,24,26)),6)
```

```
holds(1,pose(6,11,15,34),7) occurs(1,move(pose(6,11,15,34)),6)
holds(1,pose(11,15,31,34),8) occurs(1,move(pose(11,15,31,34)),7)
holds(2,pose(6,9,12,26),8) occurs(2,move(pose(6,9,12,26)),7)
holds(2,pose(6,9,11,26),9) holds(1,pose(12,15,31,34),9)
occurs(1,move(pose(12,15,31,34)),8) occurs(2,move(pose(6,9,11,26)),8)
holds(2,pose(6,9,11,34),10) holds(1,pose(12,15,31,38),10)
occurs(1,move(pose(12,15,31,38)),9) occurs(2,move(pose(6,9,11,34)),9)
holds(2,pose(6,9,15,34),11) holds(1,pose(12,31,37,38),11)
occurs(1,move(pose(12,31,37,38)),10) occurs(2,move(pose(6,9,15,34)),10)
holds(2,pose(6,10,15,34),12) holds(1,pose(12,36,37,38),12)
occurs(1,move(pose(12,36,37,38)),11) occurs(2,move(pose(6,10,15,34)),11)
holds(2,pose(10,12,15,34),13) holds(1,pose(35,36,37,38),13)
occurs(1,move(pose(35,36,37,38)),12) occurs(2,move(pose(10,12,15,34)),12)
holds(2,pose(10,12,17,34),14) holds(1,pose(35,37,38,39),14)
occurs(1,move(pose(35,37,38,39)),13) occurs(2,move(pose(10,12,17,34)),13)
holds(2,pose(10,12,17,18),15) holds(1,pose(33,37,38,39),15)
occurs(1,move(pose(33,37,38,39)),14) occurs(2,move(pose(10,12,17,18)),14)
Optimization: 1334
OPTIMUM FOUND

Models       : 4
  Optimum    : yes
Optimization : 1334
Calls        : 16
Time         : 63.326s (Solving: 1.54s 1st Model: 0.18s Unsat: 0.31s)
CPU Time     : 63.280s
```

The set of states that the first robot follow is described as follows:

$$\Sigma_0 = \Big\{(9,3,0,2),(9,8,0,2),(9,8,4,2),(28,8,4,2),(28,9,4,2),$$
$$(28,9,4,26),(28,9,6,26),(31,9,6,26),(31,34,6,26),$$
$$(35,34,6,26),(35,34,12,26),(36,34,12,26),(36,34,12,32),$$
$$(36,38,12,32),(36,38,37,32),(39,38,37,32),(39,38,37,33)\Big\}$$

The set of states that the second robot follow is described as follows:

$$\Sigma_1 = \Big\{(27,30,22,23),(27,30,24,23),(27,30,24,25),(27,31,24,25),(32,31,24,25),$$
$$(32,31,24,29),(32,12,24,29),(32,12,28,29),(37,12,28,29),$$
$$(37,12,28,31),(37,15,28,31),(37,15,11,31),(18,15,11,31),$$
$$(18,15,11,34),(18,17,11,34),(18,17,10,34),(18,17,10,15)\Big\}$$

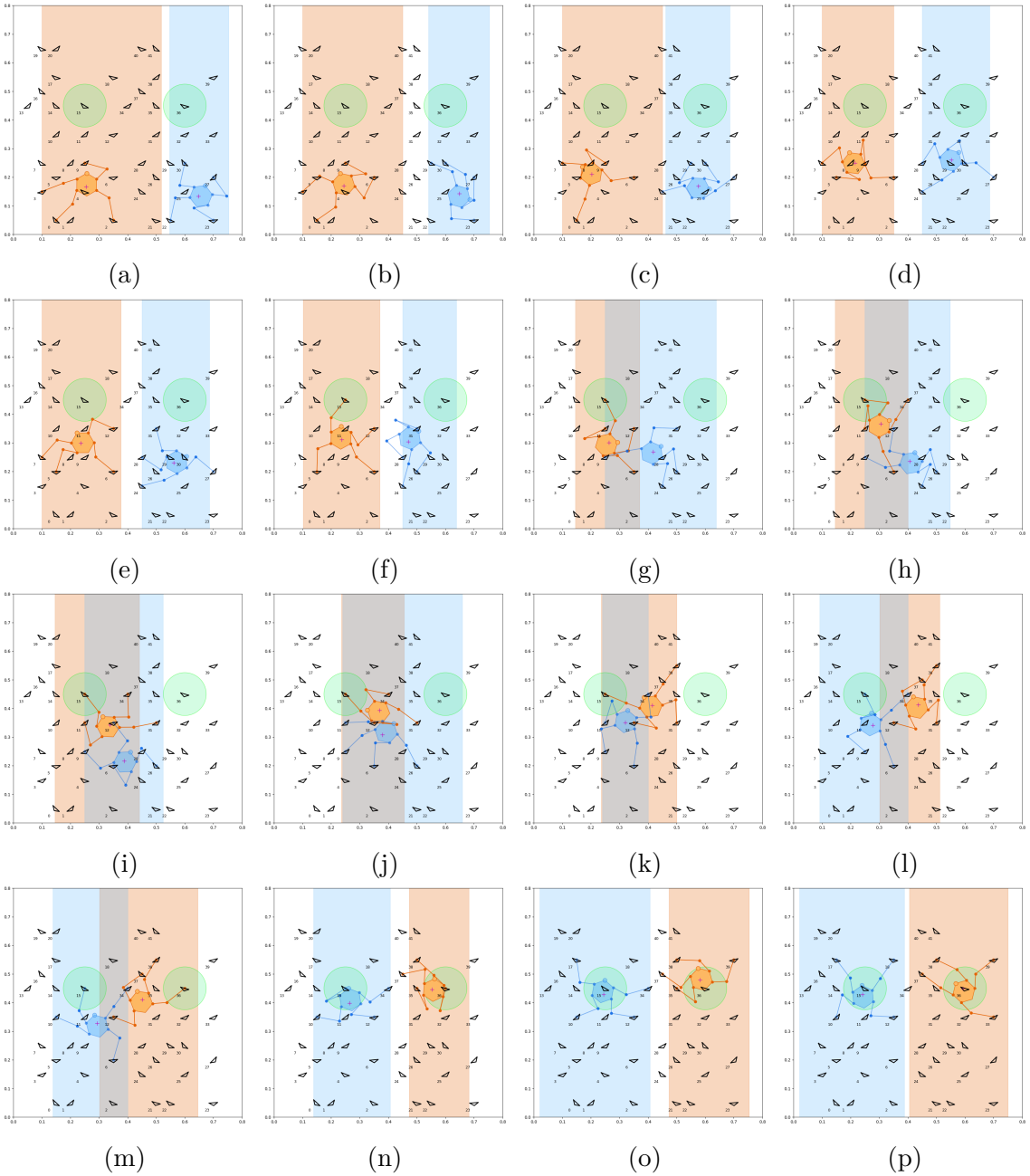


Figure 5.12 Snapshots for the task plan of the scenario with two robots

5.2 Results and Discussion

Table 5.1 summarizes the results for the sample scenarios.

- An increase in the number of holds expands the planning environment. Hence, it causes more nodes and edges, which requires more time in feasibility analysis and task planning.
- The first scenario is about horizontally climbing action. Comparing it to the second scenario performing a vertical climb, the makespan in the first scenario is larger. This is because in the first scenario, the robot plans to visit all holds, whereas in the second scenario, the planner decides to visit some holds.
- Optimum path lengths are obtained using the RRT* algorithm. The optimization function is chosen as the default function, which is the path length value. The results are proportional to the makespan because more plan time step causes more motion planning duties, creating larger path length.
- Average path lengths and planning times are average values of RRTConnect algorithm results. The standard deviation results are in parenthesis. The average path lengths are higher than in optimum ones because the optimum planner is tasked to minimize the path lengths in the desired time limit.
- The scalability of our method depends also on the size of the problem instances: more nodes and edges in feasibility graphs cause an increase in computation time spent for feasibility analysis, task planning, and motion planning.
- The example with obstacles shows that the proposed approach handles cases where there is no feasible continuous trajectory for an initial task plan, and thus the replanning loop is called to find a new task plan on the feasibility graph.

Table 5.1 Results of the sample scenarios.

Experiment Number	Hold Number	Node Number	Edge Number	Plan Time Step	State Creator Time (s)	Edge Creator Time (s)	Task Planning Time (s)	Motion Planning			Replanning Number
								Opt. Path Length	Avg. Path Length	Avg. Time (s)	
1	10	24	29	12	0.3	0.5	0.02	32.205	65.387 (6.867)	141.5 (66.6)	0
2	21	442	1937	10	7.4	43.1	1.4	30.330	55.507 (5.789)	613.0 (225.3)	0
3	42	1387	6391	15	30.5	152.3	24.4	49.187	82.860 (6.363)	688.4 (480.4)	1
4	42	1387	6391	15	30.5	152.3	50.2	46.818	72.081 (25.886)	1647.7 (3118.2)	0
5	42	1387	6391	16	30.5	152.3	126.3	-	-	-	-

6. CONCLUSIONS

We have presented a hybrid planning approach for free climbing robots that combines high-level representation and reasoning with low-level geometric reasoning, motion planning, balance, and actuator feasibility checks.

Our method first computes a feasibility graph considering balance and reachability constraints for the robot. The approach relies on a high-level reasoner to compute a sequence of poses on the feasibility graphs. The high-level reasoner enables our approach to solve optimal plans with must-be-visited poses (waypoints) must-be-avoided poses to achieve more complex tasks. Furthermore, our reasoner-based approach enables the easy extension of our method to accommodate multiple climbing robots.

Once a sequence of poses are computed, our method also calls for a sampling-based motion planner to compute collision-free continuous trajectories that ensure robot balance. Our motion planner is based on single-shot RRT-based methods and considers kinematic constraints due to the arms of the robots forming a closed-kinematic chain. This enables our motion planner to simultaneously move the body and arms of the robot to reach the desired pose, significantly improving the feasibility of motion plans.

Most importantly, our hybrid planning approach features bilateral interaction between high-level reasoning and feasibility checks. The high-level reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic/-dynamic/controls solution for that task-plan, then the feasibility checks guide the high-level reasoner by modifying the planning problem with new constraints.

Our experimental evaluations indicate that challenging instances with multiple

climbing robots, waypoints, and obstacles can be solved with the proposed hybrid planning approach. Moreover, the approach can scale well to solve large problem instances that can capture realistic problems in real life.

Future works include integration of more efficient methods for computation of feasible configurations at goal states (Dabbour, Erdem & Patoglu, 2019). It is also a tighter integration between task and motion planning such that the time available for each motion planning query is also guided by the task planner.

Extensions of our work include generalizations of the hybrid planning approach to 3D environments and holds with more complex shapes. Note that an extension to 3D environments is relatively easy, as motion planning and task planning remain the same, whereas feasibility analysis needs to be modified. These changes are concerns of low-level geometric reasoning, and they can easily be represented in 3D.

BIBLIOGRAPHY

Bagnell, J. A. & Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, (pp. 1615–1620). IEEE.

Beardsley, P., Siegwart, R., Arigoni, M., Bischoff, M., Fuhrer, S., Krummenacher, D., Mammolo, D., & Simpson, R. (2015). Vertigo – a wall-climbing robot including ground-wall transition.

Blum, T., Jones, W., & Yoshida, K. (2019). Deep learned path planning via randomized reward-linked-goals and potential space applications. *arXiv preprint arXiv:1909.06034*.

Bretl, T. (2006). Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *The International Journal of Robotics Research*, *25*(4), 317–342.

Bretl, T. W. (2005). *Multi-step motion planning: Application to free-climbing robots*. Citeseer.

Brewka, G., Eiter, T., & Truszczynski, M. (2016). Answer set programming: An introduction to the special issue. *AI Mag.*, *37*(3), 5–6.

Brockmann, W., Albrecht, S., Borrmann, D., & Elseberg, J. (2008). Dexterous energy-autarkic climbing robot. In *Advances In Mobile Robotics* (pp. 525–532). World Scientific.

Buccafurri, F., Leone, N., & Rullo, P. (2000). Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.*, *12*(5), 845–860.

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., & Patoglu, V. (2009). Bridging the gap between high-level reasoning and low-level control. In *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, (pp. 342–354).

Dabbour, A. R., Erdem, E., & Patoglu, V. (2019). Object placement on cluttered surfaces: A nested local search approach.

Daltorio, K. A., Wei, T. E., Horchler, A. D., Southard, L., Wile, G. D., Quinn, R. D., Gorb, S. N., & Ritzmann, R. E. (2009). Mini-whegs tm climbs steep surfaces using insect-inspired attachment mechanisms. *The International Journal of Robotics Research*, *28*(2), 285–302.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*.

Eich, M. & Vögele, T. (2011). Design and control of a lightweight magnetic climbing robot for vessel inspection. In *2011 19th Mediterranean Conference on Control & Automation (MED)*, (pp. 1200–1205). IEEE.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., & Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, (pp.

4575–4581).

Erdem, E., Patoglu, V., & Schüller, P. (2016). A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Communications*, *29*(2), 319–349.

Garg, U. (2018). Virtual robot climbing using reinforcement learning.

Gaschler, A., Petrick, R. P. A., Giuliani, M., Rickert, M., & Knoll, A. (2013). KVP: A knowledge of volumes approach to robot task planning. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, (pp. 202–208).

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, *19*(1), 27–82.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. (2011). Potassco: The potsdam answer set solving collection. *Ai Communications*, *24*(2), 107–124.

Gelfond, M. & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proc. of ICLP*, (pp. 1070–1080). MIT Press.

Gelfond, M. & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Gillies, S. (2020). The shapely user manual. *The Shapely User Manual - Shapely 1.7.1 documentation*.

Goldman, G. & Hong, D. (2008). Considerations for finding the optimal design parameters for a novel pole climbing robot. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 43260, (pp. 859–866).

Gravot, F., Cambon, S., & Alami, R. (2003). asymov: A planner that deals with intricate symbolic and geometric problems. In *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, (pp. 100–110).

Gullapalli, V., Franklin, J. A., & Benbrahim, H. (1994). Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine*, *14*(1), 13–24.

Hauser, K. (2008). *Motion planning for legged and humanoid robots*. Citeseer.

Hauser, K. & Latombe, J.-C. (2009). Integrating task and prm motion planning : Dealing with many infeasible motion planning queries.

Haynes, G. C., Khripin, A., Lynch, G., Amory, J., Saunders, A., Rizzi, A. A., & Koditschek, D. E. (2009). Rapid pole climbing with a quadrupedal robot. In *2009 IEEE International Conference on Robotics and Automation*, (pp. 2767–2772). IEEE.

Hertle, A., Dornhege, C., Keller, T., & Nebel, B. (2012). Planning with semantic attachments: An object-oriented view. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, (pp. 402–407).

Kaelbling, L. P. & Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *I. J. Robotics Res.*, *32*(9-10), 1194–1227.

Kavraki, L., Svestka, P., Latombe, J.-C., & Overmars, M. (1996a). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE*

Transactions on Robotics and Automation, *12*(4), 566–580.

Kavraki, L. E. & LaValle, S. M. (2016). Motion planning. In *Springer Handbook of Robotics, 2nd Ed*, Springer Handbooks (pp. 139–162). Springer.

Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (1996b). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Kennedy, B., Okon, A., Aghazarian, H., Badescu, M., Bao, X., Bar-Cohen, Y., Chang, Z., Dabiri, B. E., Garrett, M., Magnone, L., et al. (2006). Lemur iib: a robotic system for steep terrain access. *Industrial Robot: An International Journal*.

Kim, S., Spenko, M., Trujillo, S., Heyneman, B., Mattoli, V., & Cutkosky, M. R. (2007). Whole body adhesion: hierarchical, directional and distributed control of adhesive forces for a climbing robot. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, (pp. 1268–1273). IEEE.

Kingston, Z., Moll, M., & Kavraki, L. E. (2019). Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, *38*(10-11), 1151–1178.

Kingston, Z., Wells, A. M., Moll, M., & Kavraki, L. E. (2020). Informing multimodal planning with synergistic discrete leads. In *IEEE International Conference on Robotics and Automation*, (pp. 3199–3205).

Kirchner, F. (1998). Q-learning of complex behaviours on a six-legged walking machine. *Robotics and Autonomous Systems*, *25*(3-4), 253–262.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LaValle, S. M. & James J. Kuffner, J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, *20*(5), 378–400.

Lee, H., Shen, Y., Yu, C.-H., Singh, G., & Ng, A. Y. (2006). Quadruped robot obstacle negotiation via reinforcement learning. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, (pp. 3003–3010). IEEE.

Lifschitz, V. (2002a). Answer set programming and plan generation. *Artificial Intelligence*, *138*, 39–54.

Lifschitz, V. (2002b). Answer set programming and plan generation. *Artificial Intelligence*, *138*(1-2), 39–54.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Linder, S. P., Wei, E., & Clay, A. (2005). Robotic rock climbing using computer vision and force feedback. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, (pp. 4685–4690). IEEE.

Luk, B. L., Collie, A. A., & Billingsley, J. (1991). Robug ii: An intelligent wall climbing robot. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, (pp. 2342–2343). IEEE Computer Society.

Mahadevan, S. & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, *55*(2-3), 311–365.

Manoonpong, P., Parlitz, U., & Wörgötter, F. (2013). Neural control and adaptive neural forward models for insect-like, energy-efficient, and adaptable locomo-

tion of walking machines. *Frontiers in neural circuits*, *7*, 12.

Marek, V. & Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective* (pp. 375–398). Springer Verlag.

Merlet, J.-P., Gosselin, C. M., & Mouly, N. (1998). Workspaces of planar parallel manipulators. *Mechanism and Machine Theory*, *33*(1-2), 7–20.

Miller, T. G., Bretl, T. W., & Rock, S. (2006). Control of a climbing robot using real-time convex optimization. *IFAC Proceedings Volumes*, *39*(16), 409–414.

Mistry, M., Buchli, J., & Schaal, S. (2010). Inverse dynamics control of floating base systems using orthogonal decomposition. In *2010 IEEE international conference on robotics and automation*, (pp. 3406–3412). IEEE.

Nagakubo, A. & Hirose, S. (1994). Walking and running of the quadruped wall-climbing robot. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, (pp. 1005–1012). IEEE.

Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, *25*, 241–273.

Parness, A., Abcouwer, N., Fuller, C., Wiltsie, N., Nash, J., & Kennedy, B. (2017). Lemur 3: A limbed climbing robot for extreme terrain mobility in space. In *2017 IEEE international conference on robotics and automation (ICRA)*, (pp. 5467–5473). IEEE.

Plaku, E. (2012). Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Advances in Autonomous Robotics - Joint Proceedings of the 13th Annual TAROS Conference and the 15th Annual FIRA RoboWorld Congress, Bristol, UK, August 20-23, 2012*, (pp. 331–342).

Schaal, S. (1997). Learning from demonstration. *Advances in neural information processing systems*, 1040–1046.

Schilling, M., Konen, K., & Korthals, T. (2020). Modular deep reinforcement learning for emergent locomotion on a six-legged robot. In *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*, (pp. 946–953). IEEE.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, (pp. 1889–1897). PMLR.

Silva, I. J., Perico, D. H., Homem, T. P., Vilao, C. O., Tonidandel, F., & Bianchi, R. A. (2015). Using reinforcement learning to improve the stability of a humanoid robot: Walking on sloped terrain. In *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, (pp. 210–215). IEEE.

Siméon, T., Laumond, J.-P., Cortés, J., & Sahbani, A. (2004). Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, *23*(7-8), 729–746.

Simons, P., Niemelae, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artif. Intell.*, *138*(1), 181–234.

Sucan, I. A., Moll, M., & Kavraki, L. E. (2012). The open motion planning library. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82.

Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, *290*(5500), 2319–

2323.

Totani, M., Sato, N., & Morita, Y. (2019). Step climbing method for crawler type rescue robot using reinforcement learning with proximal policy optimization. In *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, (pp. 154–159). IEEE.

Tsounis, V., Alge, M., Lee, J., Farshidian, F., & Hutter, M. (2020). Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, *5*(2), 3699–3706.

Vincent, I. & Sun, Q. (2012). A combined reactive and reinforcement learning controller for an autonomous tracked vehicle. *Robotics and Autonomous Systems*, *60*(4), 599–608.

Virtanen et al., B. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, *17*(3), 261–272.

Wiley, T., Bratko, I., & Sammut, C. (2017). A machine learning system for controlling a rescue robot. In *Robot World Cup*, (pp. 108–119). Springer.

Xiao, J., Dulimarta, H., Yu, Z., Xi, N., & Tummalal, R. (2000). Dsp solution for wall-climber micro-robot control using tms320lf2407 chip. In *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems (Cat. No. CH37144)*, volume 3, (pp. 1348–1351). IEEE.

Xiao, J., Sadegh, A., Elliot, M., Calle, A., Persad, A., & Chiu, H. M. (2005). Design of mobile robots with wall climbing capability. *Proceedings of IEEE AIM, Monterey, CA*, 438–443.

Zhang, R. & Latombe, J.-C. (2013). Capuchin: A free-climbing robot. *International Journal of Advanced Robotic Systems*, *10*(4), 194.

APPENDIX A

Multi-Shot Answer Set Programming Formalism

Listing A.1 Task Planning code to be solved with CLINGO 5.4.4

```
% Incremental Clingo
#include <incmode>.

% Include the input file
#include "input.lp".

% Separate the encoding into a static part
#program base.

% Define vertices to define actions from the input edges
vertex(X) :- edge(X, _, _).
vertex(Y) :- edge(_, Y, _).

% Assign the initial case to holds
holds(C, 0) :- init(C).


% A specification of state transition, t > 0
#program step(t).

% Dynamic Programming
% 1. Actions are exogenous
% Generate an action to move a vertex from the current vertex
% through the edge.
{occurs(move(P2), t-1) : edge(P1, P2, W)} <= 1 :- holds(P1, t-1),
                                                  vertex(P1).

% 2. Direct effects of actions
% At next time step the pose will be changed if
% the robot previously at a pose and there is
% an edge connecting them and the action move/1 occurs
holds(P2, t) :- holds(P1, t-1), occurs(move(P2), t-1),
                                edge(P1, P2, W).
```

```
% 3. Preconditions of actions
% The robot arm cannot be moved to the same position where it is.
:- occurs(move(P), t-1), holds(P, t-1).

% The robot cannot move to a location where there is no edge.
:- not edge(P1, P2, _), holds(P1, t-1), occurs(move(P2), t-1).

% 4. Existence and Uniqueness of value
% At each time instant, a pose must be unique.
:- #count{P : holds(P, t)} != 1.

% 5. Inertia
{holds(P, t)} :- holds(P, t-1).

% A part for checking the goal situation and state constraints.
% t >= 0
#program check(t).

% Test
:- query(t), goal(C), not holds(C, t).
% 7. Optimize the cost of the final plan
#minimize{W : edge(P1, P2, W), holds(P1, t-1), holds(P2, t)}.


% Finally, Show the plan
#show holds / 2.
```

APPENDIX B

Case 1: Constraint equations when all legs are contacting

Constraint equations contain nonlinear equations that should satisfy the equality to the zero vector.

$$F(\mathbf{q}) = \mathbf{0} \tag{B.1}$$

The set of the constraint equations for the case where all four legs of the robot is contacting to their holds such that the body is allowed to move:

$$\begin{aligned}
x_p + r\cos(\gamma_0 + \theta_p) + l_1\cos(\gamma_0 + \theta_p + \theta_0) + l_2\cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) - x_{rf} &= 0 \\
y_p + r\sin(\gamma_0 + \theta_p) + l_1\sin(\gamma_0 + \theta_p + \theta_0) + l_2\sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) - y_{rf} &= 0 \\
x_p + r\cos(\gamma_1 + \theta_p) + l_1\cos(\gamma_1 + \theta_p + \theta_2) + l_2\cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) - x_{lf} &= 0 \\
y_p + r\sin(\gamma_1 + \theta_p) + l_1\sin(\gamma_1 + \theta_p + \theta_2) + l_2\sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) - y_{lf} &= 0 \\
x_p + r\cos(\gamma_2 + \theta_p) + l_1\cos(\gamma_2 + \theta_p + \theta_4) + l_2\cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) - x_{lr} &= 0 \\
y_p + r\sin(\gamma_2 + \theta_p) + l_1\sin(\gamma_2 + \theta_p + \theta_4) + l_2\sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) - y_{lr} &= 0 \\
x_p + r\cos(\gamma_3 + \theta_p) + l_1\cos(\gamma_3 + \theta_p + \theta_6) + l_2\cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) - x_{rr} &= 0 \\
y_p + r\sin(\gamma_3 + \theta_p) + l_1\sin(\gamma_3 + \theta_p + \theta_6) + l_2\sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) - y_{rr} &= 0
\end{aligned} \tag{B.2}$$

The related Jacobian matrix can be derived by taking the partial derivative of the function $F(\mathbf{q})$:

$$\mathbf{J} = \frac{\partial F}{\partial \mathbf{q}} \tag{B.3}$$

that is the following matrix format:

$$\mathbf{J} = \begin{bmatrix} j_{1,1} & j_{1,2} & 0 & \cdots & \cdots & 0 & 1 & 0 & j_{1,11} \\ j_{2,1} & j_{2,2} & \ddots & \ddots & & \vdots & 0 & 1 & j_{2,11} \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 & \vdots & \vdots & \vdots \\ \vdots & & \ddots & \ddots & j_{7,7} & j_{7,8} & 1 & 0 & j_{7,11} \\ 0 & \cdots & \cdots & 0 & j_{8,7} & j_{8,8} & 0 & 1 & j_{8,11} \end{bmatrix} \tag{B.4}$$

where

$$
\begin{aligned}
j_{1,1} &= -l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,2} &= -l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,11} &= -r \sin(\gamma_0 + \theta_p) - l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{2,1} &= l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,2} &= l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,11} &= r \cos(\gamma_0 + \theta_p) + l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{3,3} &= -l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,4} &= -l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,11} &= -r \sin(\gamma_1 + \theta_p) - l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{4,3} &= l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,4} &= l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,11} &= r \cos(\gamma_1 + \theta_p) + l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{5,5} &= -l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{5,6} &= -l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{5,11} &= -r \sin(\gamma_2 + \theta_p) - l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{6,5} &= l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{6,6} &= l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{6,11} &= r \cos(\gamma_2 + \theta_p) + l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{7,7} &= -l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{7,8} &= -l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{7,11} &= -r \sin(\gamma_3 + \theta_p) - l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

$$
\begin{aligned}
j_{8,7} &= l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{8,8} &= l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{8,11} &= r \cos(\gamma_3 + \theta_p) + l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

Case 2: Constraint equations when all legs are contacting except the right front leg

Constraint equations contain nonlinear equations that should satisfy the equality to the zero vector.

$$F(\mathbf{q}) = \mathbf{0} \tag{B.5}$$

The set of the constraint equations for the case where three legs of the robot are contacting to their holds such that the body and the right front leg are allowed to move:

$$\begin{aligned}
x_p + r\cos(\gamma_1 + \theta_p) + l_1\cos(\gamma_1 + \theta_p + \theta_2) + l_2\cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) - x_{lf} = 0 \\
y_p + r\sin(\gamma_1 + \theta_p) + l_1\sin(\gamma_1 + \theta_p + \theta_2) + l_2\sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) - y_{lf} = 0 \\
x_p + r\cos(\gamma_2 + \theta_p) + l_1\cos(\gamma_2 + \theta_p + \theta_4) + l_2\cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) - x_{lr} = 0 \\
y_p + r\sin(\gamma_2 + \theta_p) + l_1\sin(\gamma_2 + \theta_p + \theta_4) + l_2\sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) - y_{lr} = 0 \\
x_p + r\cos(\gamma_3 + \theta_p) + l_1\cos(\gamma_3 + \theta_p + \theta_6) + l_2\cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) - x_{rr} = 0 \\
y_p + r\sin(\gamma_3 + \theta_p) + l_1\sin(\gamma_3 + \theta_p + \theta_6) + l_2\sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) - y_{rr} = 0
\end{aligned} \tag{B.6}$$

The related Jacobian matrix can be derived by taking the partial derivative of the function $F(\mathbf{q})$:

$$\mathbf{J} = \frac{\partial F}{\partial \mathbf{q}} \tag{B.7}$$

that is the following matrix format:

$$\mathbf{J} = \begin{bmatrix}
0 & 0 & j_{1,3} & j_{1,4} & 0 & 0 & 0 & 0 & 1 & 0 & j_{1,11} \\
0 & 0 & j_{2,3} & j_{2,4} & 0 & 0 & 0 & 0 & 0 & 1 & j_{2,11} \\
0 & 0 & 0 & 0 & j_{3,5} & j_{3,6} & 0 & 0 & 1 & 0 & j_{3,11} \\
0 & 0 & 0 & 0 & j_{4,5} & j_{4,6} & 0 & 0 & 0 & 1 & j_{4,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{5,7} & j_{5,8} & 1 & 0 & j_{5,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{6,7} & j_{6,8} & 0 & 1 & j_{6,11}
\end{bmatrix} \tag{B.8}$$

where

$$
\begin{aligned}
j_{1,3} &= -l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{1,4} &= -l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{1,11} &= -r \sin(\gamma_1 + \theta_p) - l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{2,3} &= l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{2,4} &= l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{2,11} &= r \cos(\gamma_1 + \theta_p) + l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{3,5} &= -l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{3,6} &= -l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{3,11} &= -r \sin(\gamma_2 + \theta_p) - l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{4,5} &= l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{4,6} &= l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{4,11} &= r \cos(\gamma_2 + \theta_p) + l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{5,7} &= -l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,8} &= -l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,11} &= -r \sin(\gamma_3 + \theta_p) - l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

$$
\begin{aligned}
j_{6,7} &= l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,8} &= l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,11} &= r \cos(\gamma_3 + \theta_p) + l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

Case 3: Constraint equations when all legs are contacting except the left front leg

Constraint equations contain nonlinear equations that should satisfy the equality to the zero vector.

$$F(\mathbf{q}) = \mathbf{0} \tag{B.9}$$

The set of the constraint equations for the case where three legs of the robot are contacting to their holds such that the body and the left front leg are allowed to move:

$$\begin{aligned}
x_p + r\cos(\gamma_0 + \theta_p) + l_1\cos(\gamma_0 + \theta_p + \theta_0) + l_2\cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) - x_{rf} = 0 \\
y_p + r\sin(\gamma_0 + \theta_p) + l_1\sin(\gamma_0 + \theta_p + \theta_0) + l_2\sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) - y_{rf} = 0 \\
x_p + r\cos(\gamma_2 + \theta_p) + l_1\cos(\gamma_2 + \theta_p + \theta_4) + l_2\cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) - x_{lr} = 0 \\
y_p + r\sin(\gamma_2 + \theta_p) + l_1\sin(\gamma_2 + \theta_p + \theta_4) + l_2\sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) - y_{lr} = 0 \\
x_p + r\cos(\gamma_3 + \theta_p) + l_1\cos(\gamma_3 + \theta_p + \theta_6) + l_2\cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) - x_{rr} = 0 \\
y_p + r\sin(\gamma_3 + \theta_p) + l_1\sin(\gamma_3 + \theta_p + \theta_6) + l_2\sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) - y_{rr} = 0
\end{aligned} \tag{B.10}$$

The related Jacobian matrix can be derived by taking the partial derivative of the function $F(\mathbf{q})$:

$$\mathbf{J} = \frac{\partial F}{\partial \mathbf{q}} \tag{B.11}$$

that is the following matrix format:

$$\mathbf{J} = \begin{bmatrix}
j_{1,1} & j_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & j_{1,11} \\
j_{2,2} & j_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & j_{2,11} \\
0 & 0 & 0 & 0 & j_{3,5} & j_{3,6} & 0 & 0 & 1 & 0 & j_{3,11} \\
0 & 0 & 0 & 0 & j_{4,5} & j_{4,6} & 0 & 0 & 0 & 1 & j_{4,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{5,7} & j_{5,8} & 1 & 0 & j_{5,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{6,7} & j_{6,8} & 0 & 1 & j_{6,11}
\end{bmatrix} \tag{B.12}$$

where

$$
\begin{aligned}
j_{1,1} &= -l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,2} &= -l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,11} &= -r \sin(\gamma_0 + \theta_p) - l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{2,1} &= l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,2} &= l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,11} &= r \cos(\gamma_0 + \theta_p) + l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{3,5} &= -l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{3,6} &= -l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{3,11} &= -r \sin(\gamma_2 + \theta_p) - l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{4,5} &= l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{4,6} &= l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{4,11} &= r \cos(\gamma_2 + \theta_p) + l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{5,7} &= -l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,8} &= -l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,11} &= -r \sin(\gamma_3 + \theta_p) - l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

$$
\begin{aligned}
j_{6,7} &= l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,8} &= l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,11} &= r \cos(\gamma_3 + \theta_p) + l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

Case 4: Constraint equations when all legs are contacting except the left rear leg

Constraint equations contain nonlinear equations that should satisfy the equality to the zero vector.

$$F(\mathbf{q}) = \mathbf{0} \tag{B.13}$$

The set of the constraint equations for the case where three legs of the robot are contacting to their holds such that the body and the left rear leg are allowed to move:

$$\begin{aligned}
x_p + r\cos(\gamma_0 + \theta_p) + l_1\cos(\gamma_0 + \theta_p + \theta_0) + l_2\cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) - x_{rf} = 0 \\
y_p + r\sin(\gamma_0 + \theta_p) + l_1\sin(\gamma_0 + \theta_p + \theta_0) + l_2\sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) - y_{rf} = 0 \\
x_p + r\cos(\gamma_1 + \theta_p) + l_1\cos(\gamma_1 + \theta_p + \theta_2) + l_2\cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) - x_{lf} = 0 \\
y_p + r\sin(\gamma_1 + \theta_p) + l_1\sin(\gamma_1 + \theta_p + \theta_2) + l_2\sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) - y_{lf} = 0 \\
x_p + r\cos(\gamma_3 + \theta_p) + l_1\cos(\gamma_3 + \theta_p + \theta_6) + l_2\cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) - x_{rr} = 0 \\
y_p + r\sin(\gamma_3 + \theta_p) + l_1\sin(\gamma_3 + \theta_p + \theta_6) + l_2\sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) - y_{rr} = 0
\end{aligned} \tag{B.14}$$

The related Jacobian matrix can be derived by taking the partial derivative of the function $F(\mathbf{q})$:

$$\mathbf{J} = \frac{\partial F}{\partial \mathbf{q}} \tag{B.15}$$

that is the following matrix format:

$$\mathbf{J} = \begin{bmatrix}
j_{1,1} & j_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & j_{1,11} \\
j_{2,2} & j_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & j_{2,11} \\
0 & 0 & j_{3,3} & j_{3,4} & 0 & 0 & 0 & 0 & 1 & 0 & j_{3,11} \\
0 & 0 & j_{4,3} & j_{4,4} & 0 & 0 & 0 & 0 & 0 & 1 & j_{4,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{5,7} & j_{5,8} & 1 & 0 & j_{5,11} \\
0 & 0 & 0 & 0 & 0 & 0 & j_{6,7} & j_{6,8} & 0 & 1 & j_{6,11}
\end{bmatrix} \tag{B.16}$$

where

$$
\begin{aligned}
j_{1,1} &= -l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,2} &= -l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,11} &= -r \sin(\gamma_0 + \theta_p) - l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{2,1} &= l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,2} &= l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,11} &= r \cos(\gamma_0 + \theta_p) + l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{3,3} &= -l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,4} &= -l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,11} &= -r \sin(\gamma_1 + \theta_p) - l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{4,3} &= l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,4} &= l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,11} &= r \cos(\gamma_1 + \theta_p) + l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{5,7} &= -l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,8} &= -l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{5,11} &= -r \sin(\gamma_3 + \theta_p) - l_1 \sin(\gamma_3 + \theta_p + \theta_6) - l_2 \sin(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

$$
\begin{aligned}
j_{6,7} &= l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,8} &= l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7) \\
j_{6,11} &= r \cos(\gamma_3 + \theta_p) + l_1 \cos(\gamma_3 + \theta_p + \theta_6) + l_2 \cos(\gamma_3 + \theta_p + \theta_6 + \theta_7)
\end{aligned}
$$

Case 5: Constraint equations when all legs are contacting except the right rear leg

Constraint equations contain nonlinear equations that should satisfy the equality to the zero vector.

$$F(\mathbf{q}) = \mathbf{0} \tag{B.17}$$

The set of the constraint equations for the case where three legs of the robot are contacting to their holds such that the body and the right rear leg are allowed to move:

$$\begin{aligned}
x_p + r\cos(\gamma_0 + \theta_p) + l_1\cos(\gamma_0 + \theta_p + \theta_0) + l_2\cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) - x_{rf} = 0 \\
y_p + r\sin(\gamma_0 + \theta_p) + l_1\sin(\gamma_0 + \theta_p + \theta_0) + l_2\sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) - y_{rf} = 0 \\
x_p + r\cos(\gamma_1 + \theta_p) + l_1\cos(\gamma_1 + \theta_p + \theta_2) + l_2\cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) - x_{lf} = 0 \\
y_p + r\sin(\gamma_1 + \theta_p) + l_1\sin(\gamma_1 + \theta_p + \theta_2) + l_2\sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) - y_{lf} = 0 \\
x_p + r\cos(\gamma_2 + \theta_p) + l_1\cos(\gamma_2 + \theta_p + \theta_4) + l_2\cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) - x_{lr} = 0 \\
y_p + r\sin(\gamma_2 + \theta_p) + l_1\sin(\gamma_2 + \theta_p + \theta_4) + l_2\sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) - y_{lr} = 0
\end{aligned} \tag{B.18}$$

The related Jacobian matrix can be derived by taking the partial derivative of the function $F(\mathbf{q})$:

$$\mathbf{J} = \frac{\partial F}{\partial \mathbf{q}} \tag{B.19}$$

that is the following matrix format:

$$\mathbf{J} = \begin{bmatrix}
j_{1,1} & j_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & j_{1,11} \\
j_{2,2} & j_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & j_{2,11} \\
0 & 0 & j_{3,3} & j_{3,4} & 0 & 0 & 0 & 0 & 1 & 0 & j_{3,11} \\
0 & 0 & j_{4,3} & j_{4,4} & 0 & 0 & 0 & 0 & 0 & 1 & j_{4,11} \\
0 & 0 & 0 & 0 & j_{5,7} & j_{5,8} & 0 & 0 & 1 & 0 & j_{5,11} \\
0 & 0 & 0 & 0 & j_{6,7} & j_{6,8} & 0 & 0 & 0 & 1 & j_{6,11}
\end{bmatrix} \tag{B.20}$$

where

$$
\begin{aligned}
j_{1,1} &= -l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,2} &= -l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{1,11} &= -r \sin(\gamma_0 + \theta_p) - l_1 \sin(\gamma_0 + \theta_p + \theta_0) - l_2 \sin(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{2,1} &= l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,2} &= l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1) \\
j_{2,11} &= r \cos(\gamma_0 + \theta_p) + l_1 \cos(\gamma_0 + \theta_p + \theta_0) + l_2 \cos(\gamma_0 + \theta_p + \theta_0 + \theta_1)
\end{aligned}
$$

$$
\begin{aligned}
j_{3,3} &= -l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,4} &= -l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{3,11} &= -r \sin(\gamma_1 + \theta_p) - l_1 \sin(\gamma_1 + \theta_p + \theta_2) - l_2 \sin(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{4,3} &= l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,4} &= l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3) \\
j_{4,11} &= r \cos(\gamma_1 + \theta_p) + l_1 \cos(\gamma_1 + \theta_p + \theta_2) + l_2 \cos(\gamma_1 + \theta_p + \theta_2 + \theta_3)
\end{aligned}
$$

$$
\begin{aligned}
j_{5,5} &= -l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{5,6} &= -l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{5,11} &= -r \sin(\gamma_2 + \theta_p) - l_1 \sin(\gamma_2 + \theta_p + \theta_4) - l_2 \sin(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$

$$
\begin{aligned}
j_{6,5} &= l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{6,6} &= l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5) \\
j_{6,11} &= r \cos(\gamma_2 + \theta_p) + l_1 \cos(\gamma_2 + \theta_p + \theta_4) + l_2 \cos(\gamma_2 + \theta_p + \theta_4 + \theta_5)
\end{aligned}
$$