Efficient Quadruped Mobility: Harnessing a Generalist Policy for Streamlined Planning

Sayan Mondal

CMU-RI-TR-24-72

December 18, 2024

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Dimitrios Apostolopoulos , *Chair*
Prof. Maxim Likhachev
Rishi Veerapaneni

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my Mom, Dad, and Nawab Mama.

Abstract

Navigating quadruped robots through complex, unstructured environments over long horizons remains a critical challenge in robotics. Traditional planning methods excel in providing guarantees such as optimality and long-horizon reasoning, while learning-based approaches, particularly those leveraging Deep Reinforcement Learning (DRL), offer robustness and adaptability. This thesis introduces **S3D-OWNS (Skilled 3D-Optimal Waypoint Navigation System)**, a novel hybrid framework that combines the strengths of both paradigms to achieve efficient and adaptive quadrupedal locomotion.

The S3D-OWNS framework integrates a high-level sampling-based planner with a generalist DRL-trained locomotion policy. The planner handles long-horizon navigation and optimal path planning by reasoning over obstacle traversability, while the learned policy executes agile, real-time locomotion tasks such as walking, jumping, and climbing. By leveraging DRL-trained policies, the dimensionality of the planning state space is reduced, enabling computational efficiency and allowing the locomotion policy to manage complex maneuvers. This integration empowers the system to navigate cluttered environments while optimizing for energy consumption, time efficiency, and task success.

Key innovations of this work include:

- A goal-conditioned locomotion policy trained across diverse terrains using DRL, ensuring robustness and adaptability.
- A sampling-based planner that evaluates obstacle traversability based on the robot's capabilities, enabling efficient path planning beyond traditional obstacle avoidance.
- Cost predictors trained using GPU parallelization to estimate energy expenditure, traversal time, and success likelihood for each path segment.
- A modular design that simplifies heading control by implicitly aligning the robot's orientation through waypoint placement.

Extensive experimentation in simulated environments with a Unitree Go1 quadruped demonstrates that S3D-OWNS significantly outperforms traditional collision-avoidance planners in navigation efficiency. The system optimally utilizes the robot's climbing and jumping skills to reduce energy consumption or traversal time across challenging terrains. Ablation

studies validate the contributions of individual components, highlighting improvements in operational efficiency and task success rates.

This research advances quadrupedal robotics by showcasing how hybrid systems can combine classical planning with modern AI-driven techniques to achieve both optimality and adaptability. The scalability of S3D-OWNS across different robot models and sensor configurations makes it applicable to diverse domains such as industrial automation, search-and-rescue missions, and exploration in unstructured environments. By addressing key challenges in long-horizon navigation and dynamic terrain adaptation, this work sets a foundation for more efficient and versatile robotic systems.

Acknowledgments

First and foremost, I wish to express my heartfelt gratitude to Prof. Maxim Likhachev for his unwavering support, mentorship, and encouragement throughout my Master's journey at CMU. Under his guidance, I had the invaluable opportunity to work on a research project that resonated deeply with my interests, with full autonomy to shape my own research direction. Prof. Likhachev fostered an environment of trust and openness, making it possible for me to explore, challenge, and grow as a researcher. His consistent guidance, coupled with his kindness, provided the perfect balance of independence and support, allowing me to develop confidence in my work. His Planning and Decision-Making in Robotics class and our weekly project discussions were not only insightful but transformative in my learning journey. I am profoundly thankful for the platform he provided, where I could openly share my ideas, ask questions, and pursue the path I truly aspired to.

I am also immensely grateful to Prof. Dimitrios (Dimi) Apostolopoulos, my advisor, for his steadfast support throughout the MSR program. His understanding and compassionate approach helped me during challenging times, both in research and in logistical matters. His willingness to let me pivot my research when I felt a need for growth reflects his remarkable qualities as a mentor and leader, always placing his students' well-being first. I am especially thankful for his generosity in providing an office space and a dedicated system for my work, which played an instrumental role in my research progress. I could not have asked for a more supportive advisor. Thank you sincerely, Prof. Apostolopoulos, for your time, guidance, and kindness.

My sincere thanks go to Rishi Veerapaneni, who was always there whenever I encountered roadblocks in my research. His insights, especially during brainstorming sessions, proved invaluable, and his willingness to help me troubleshoot my code speaks to both his expertise and generosity. He is not only one of the brightest minds I've met at CMU but also one of the kindest.

I also extend my gratitude to my collaborators, Zhikai (Logan) Zhang and Siddharth Saha. Together, we shared many late nights, engaged in stimulating discussions, and enjoyed a unique camaraderie. The mutual learning and enjoyment from our collaboration made this project particularly fulfilling, and I am fortunate to have been part of such an outstanding team.

Finally, I want to express my deepest gratitude to my family. To my parents, who have been a source of unwavering support throughout this journey, and to my sister, who has always believed in me and filled my life with color. Your encouragement has been my greatest motivation, and I am forever grateful.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

Chapter 1

Introduction

1.0.1 Motivation

Recent advancements in simulators like Isaac Gym [6] have revolutionized the training of Deep Reinforcement Learning (DRL) locomotion policies. These high-performance simulators leverage GPU parallelization to accelerate learning, enabling the development of robust and generalizable policies across diverse terrains in a matter of days. The rapid progress in this field has led to the emergence of policies that can handle complex terrains, dynamic obstacles, and even recover from falls, significantly expanding the operational capabilities of legged robots [1][3][15]. A library of low-level skills equips such robots with a versatile set of options for navigating various environments. Building on this foundation, our work focuses on designing a framework that can effectively guide these agile robots, ensuring both efficiency and safety in navigation.

1.0.2 Problem Statement

Efficient mobility in legged robots is deeply inspired by biological systems, where high-level cognitive centers provide strategic commands while lower centers execute precise motor actions. This hierarchical framework, often described as the dual-process theory, is analogous to System I and System II in robotics. System II handles high-level decision-making and planning, while System I performs reflexive, fast, and intuitive actions through learned locomotion skills. This biomimetic design enhances

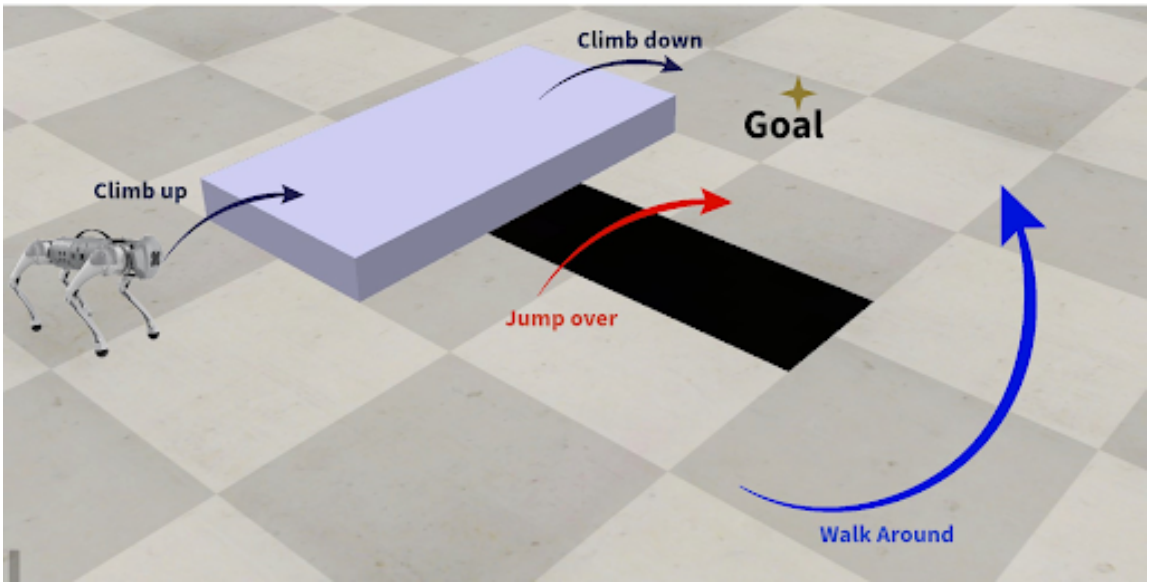


Figure 1.1: Problem statement: Given a cluttered environment, a robot needs to decide which path to take to reach a destination, based on certain user-objectives: minimum energy or minimum time above a certain risk threshold. Users can thus expand the capabilities of legged systems having multiple skills for navigation based on their need.

both the efficiency of robotic locomotion and its adaptability to diverse and dynamic environments, which is crucial for real-world applications.

The synergy between strategic planning and deep reinforcement learning (DRL)-based locomotion policies enables robots to adapt to complex terrains and efficiently navigate cluttered 3D environments. Much like animals instinctively adjust their movements to their surroundings, our approach combines a high-level planner with a generalist locomotion policy capable of executing a diverse set of skills. This integration bridges the gap between traditional robotics and modern AI techniques, forming a hybrid system that leverages the strengths of both paradigms to achieve efficient quadruped mobility in challenging environments.

In industrial settings such as warehouses and manufacturing units, the deployment of robotic systems requires a careful balance of safety, optimality, and cost-effectiveness. Robots must complete long-horizon tasks, navigating intricate static environments while avoiding obstacles with precision, all while operating efficiently over extended

periods with minimal recharging. These constraints present significant challenges, necessitating systems that can reliably plan and execute actions in complex, obstacle-laden terrains.

The potential applications of efficient quadruped mobility extend well beyond industrial contexts. In disaster response scenarios, legged robots could traverse unstable or debris-filled terrains to assist in search and rescue operations. Similarly, in agriculture, quadruped robots could aid in tasks like crop monitoring and harvesting across diverse and challenging landscapes. By addressing these diverse use cases, our approach aims to develop robust, efficient, and adaptable robotic systems for a wide range of applications.

1.0.3 Contributions

Our research builds upon recent advancements in both planning algorithms and reinforcement learning. Notable works in this field include the development of hierarchical planning frameworks that decompose complex tasks into manageable sub-goals and the creation of terrain-aware locomotion policies that can adapt to different surface properties in real-time.

By combining a high-level planner with a generalist locomotion policy, our approach aims to address the limitations of current systems, which often struggle to scale across diverse terrains or handle long-horizon tasks for multi-skilled robots. This integration not only improves the robot's ability to navigate complex environments but also enhances its energy efficiency, a critical factor for prolonged operation in industrial settings.

The development of such advanced mobility systems for quadruped robots represents a significant step towards more versatile and autonomous robotic platforms. As these technologies mature, they have the potential to revolutionize various industries, from logistics and manufacturing to exploration and environmental monitoring.

Legged robots, particularly quadrupeds, excel in areas where traditional wheeled robots struggle, such as environments filled with gaps, steps, and clutter. Quadrupeds offer significant advantages over humanoids in terms of stability, simplicity (fewer degrees of freedom), and cost-effectiveness, making them ideal candidates for scalable industrial deployment. These advantages stem from their biomimetic design, which

allows for superior adaptability to uneven terrain and improved energy efficiency during locomotion. To meet industry demands, these robots must optimize energy consumption, ensure safety, and maintain high task completion rates. This necessitates sophisticated long-horizon planning for smooth, uninterrupted operation. Our work focuses on harnessing these advantages of quadrupeds while addressing the critical need for efficient, safe, and cost-effective robotic solutions in industrial settings.

The application of quadruped robots in industrial environments presents unique challenges, including the need for precise navigation in dynamic spaces, real-time obstacle avoidance, and seamless integration with existing workflows. Additionally, these robots must operate within strict safety parameters to ensure the well-being of human workers in shared spaces.

Current approaches to robotic navigation often struggle to balance efficiency, safety, and adaptability across diverse terrains. Many systems either lack the flexibility to handle varied environments or fall short in managing long-horizon tasks for multi-skilled robots. This limitation is particularly evident in scenarios requiring complex decision-making, such as navigating through cluttered areas or selecting optimal paths in multi-level environments.

To address these limitations, we present the Skilled 3D - Optimal Waypoint Navigation System (S3D-OWNS), a novel framework that integrates a simple yet effective planner (System II) with a generalist reinforcement learning-based locomotion policy (System I). S3D-OWNS is designed to handle the complexities of long-horizon navigation while maintaining optimality in terms of energy, risk, and time. This dual system approach allows for real-time adaptation to environmental changes while maintaining a high-level strategic overview of the navigation task.

Our system enables legged robots to efficiently navigate unstructured and cluttered environments. The key contributions of our work are as follows:

- **Flexible Planning for Customizable Behavior:** We propose a planning system capable of optimizing paths based on user-defined priorities, such as minimizing risk, maximizing energy efficiency, or achieving a balance between the two. This flexibility allows robots to adapt their behavior to specific industrial scenarios, such as prioritizing safety in hazardous areas or maximizing speed and efficiency in time-sensitive operations.

- **Nonholonomic Navigation for Enhanced Safety:** Our approach accounts for the robot's physical constraints and sensor limitations, ensuring safe operation by avoiding blind spots. By explicitly integrating these constraints, the risk of collisions is significantly reduced.
- **Constraint-Aware Planning:** We integrate the robot's capabilities and limitations, such as motor torque, and sensor range, into the planning process. This ensures that all computed paths are within the robot's operational capabilities, improving reliability and robustness in real-world deployments.
- **Reduced State Space for Efficient Planning:** The planner operates in a reduced state space by implicitly controlling the robot's heading direction through waypoint planning. The locomotion policy ensures that the robot always faces the direction of the line connecting the current waypoint to the next, eliminating the need for explicit heading adjustments. This simplification improves computational efficiency and ensures smoother transitions between waypoints as well as preventing the robot to take very sharp turns that could cause instability.
- **Improved Planning Efficiency and Optimality:** Leveraging a novel combination of heuristic-based path planning and reinforcement learning-based locomotion policies, our system achieves significant improvements in both computation time and solution quality compared to obstacle-avoidance based planners that does not leverage the robot's capabilities. This hybrid approach addresses the limitations of traditional planners, which often struggle with the complexity of dynamic environments.

Additionally, our framework incorporates advanced features that expand its versatility:

- **Multi-Modal Locomotion:** Our generalist locomotion policy enables seamless transitions between different gaits—such as walking, climbing, and jumping—based on terrain and task requirements. This adaptability significantly enhances navigation in diverse environments, outperforming systems limited to fixed gaits. This generalist locomotion policy is inspired by the work of Cheng et al. [1].
- **Scalability Across Platforms:** The framework is designed to support different

quadruped models and sensor configurations, making it applicable to a wide range of industrial and operational scenarios. This scalability positions our framework as a versatile tool for various robotic applications.

Compared to prior work, which often focuses either on efficient planning or robust locomotion, our approach combines these strengths into a unified system. This integration not only improves navigation performance but also bridges the gap between traditional robotic planning methods and modern AI-driven locomotion techniques.

Using a Unitree Go1 quadruped equipped with a single front-facing depth camera, we showcase S3D-OWNS' effectiveness in simulated environments featuring challenging obstacles like blocks and gaps. The choice of the Unitree Go1 is significant due to its balance of affordability and capability, making it a representative model for potential large-scale industrial deployment. Our approach not only addresses current limitations in robotic navigation but also paves the way for more efficient and adaptable robotic systems in industrial applications.

The implications of this research extend beyond immediate industrial applications. The principles developed in S3D-OWNS could be applied to other domains such as search and rescue operations, planetary exploration, and even assistive robotics. By combining efficient planning with adaptive locomotion, our work contributes to the broader goal of creating more autonomous and versatile robotic systems capable of operating in diverse and challenging environments.

Chapter 2

Related Work

2.0.1 Policy

Recent advances in deep reinforcement learning have revolutionized quadruped locomotion control. The development of simulators like Isaac Gym has enabled training of robust and generalizable policies within days through GPU parallelization [9]. These policies have demonstrated remarkable capabilities ranging from basic locomotion to complex maneuvers like parkour and fall recovery. The Extreme Parkour framework showed how a single neural network policy operating directly from camera images can enable precise athletic behaviors on low-cost robots with imprecise actuation, achieving jumps up to 2x the robot's height and length [1]. The ANYmal Parkour framework demonstrates how a hierarchical learning approach can enable agile navigation across challenging parkour-like scenarios without requiring expert demonstrations or explicit contact modeling [3]. This system achieves impressive speeds of up to 2 meters per second while navigating consecutive challenging obstacles, showcasing the potential of end-to-end learned controllers. Modern approaches to quadruped control often employ hierarchical frameworks to manage complexity. The ANYmal system uses a hierarchical structure where a high-level policy selects and controls specialized locomotion skills (walking, jumping, climbing, and crouching) based on terrain understanding. This architecture allows the navigation policy to be aware of each skill's capabilities and adapt behavior accordingly, enabling more robust performance across diverse scenarios. End-to-end learning approaches have shown

promising results in directly mapping sensory inputs to control actions. Recent work demonstrates how neural networks can reconstruct obstacles from highly occluded and noisy sensory data, enabling scene understanding for navigation. The Extreme Parkour framework shows how a single neural network can process depth camera inputs to generate precise control outputs, even with imperfect sensing and actuation.

A key challenge in learning-based approaches is transferring policies trained in simulation to real hardware. The ANYmal Parkour system successfully demonstrates sim-to-real transfer, enabling the robot to navigate challenging obstacles in the real world despite being trained entirely in simulation [3]. This achievement highlights the importance of robust simulation environments and effective domain randomization techniques.

2.0.2 Cost Predictor

Traditional approaches to quadruped navigation often rely on elevation maps constructed by fusing point cloud and odometry data. However, Yang et al. demonstrated that a GPU-aided, sampling-based path planner combined with a gradient-based optimizer can achieve planning speeds three orders of magnitude faster than RRT* while maintaining optimality [13]. Their framework leverages a neural network-based locomotion cost predictor trained in simulation to understand the robot's capabilities across different terrain types.

Recent work has shown the effectiveness of learning-based approaches for predicting motion costs. Yang et al.'s framework uses a CNN-based cost predictor that estimates multiple motion attributes based on local height scans and locomotion commands [13]. This enables real-time path optimization that considers both terrain characteristics and robot capabilities.

2.0.3 Planner

The iPlanner framework demonstrates how end-to-end learning using only front-facing depth images can enable efficient local path planning [14]. This approach achieves around $4\times$ faster planning than classic methods while maintaining robustness against localization noise. Similarly, ViPlanner extends this concept by incorporating

semantic information, reducing traversability costs by 38.02% compared to purely geometric approaches [8].

A novel direction in path planning employs diffusion models for trajectory generation. The DiPPeST framework shows how diffusion-based planning can achieve a 92% success rate in obstacle avoidance for nominal environments and an 88% success rate in complex scenarios [11] . This approach enables zero-shot adaptation and real-time path refinements reactive to camera input, without requiring additional training or environment interpretation techniques.

Advanced planning strategies now consider the unique capabilities of legged robots. The reachability-based navigation planner proposed by ETH Zurich approximates robot morphology using reachability and body volumes [12] . This approach enables real-time performance with fast update rates even in cluttered and narrow environments by validating only low-cost graph edges during graph expansion and implementing an adaptive sampling scheme.

The integration of semantic information has emerged as a crucial component in modern navigation systems. ViPlanner's approach of using 30 semantic classes encoded in RGB colorspace enables effective representation of multiple traversability levels [8]. This semantic understanding allows robots to distinguish between different terrain types and accurately identify obstacles, leading to more efficient navigation strategies.

Recent frameworks demonstrate impressive sim-to-real transfer capabilities. The iPlanner achieves robust performance in real-world scenarios without requiring real-world training data [14]. Similarly, reachability-based planners have shown successful deployment on quadrupedal robots like ANYmal, demonstrating their effectiveness in practical applications [12].

Modern planning frameworks increasingly combine multiple approaches to leverage their respective strengths. The integration of geometric planning with semantic understanding, as demonstrated by ViPlanner [8] , and the combination of sampling-based planning with gradient-based optimization [13] represent this trend toward multi-modal solutions that can handle diverse environmental challenges.

Chapter 3

Approach

3.1 Preliminaries

Motivation of Our Framework

Efficient navigation for quadruped robots poses several challenges, including reasoning over long horizons, adapting to diverse and dynamic terrains, and generalizing to unseen scenarios. Addressing these challenges requires a careful balance of planning and execution capabilities. Inspired by insights into human cognition, we draw upon psychologist Daniel Kahneman's dual-process theory, as introduced in *Thinking, Fast and Slow* [5]. Kahneman describes two modes of thought:

- **System 1**, which is fast, instinctive, and intuitive, is used for tasks that have been practiced to the point of becoming second nature. This mode prioritizes speed and efficiency, even at the cost of potential risk.
- **System 2**, which is slower, deliberative, and logical, is used for tasks requiring careful reasoning and attention to variation, particularly for complex or unfamiliar situations.

In our work, we model this paradigm by separating tasks between a high-level planner, which mimics the reasoning of "System 2," and a low-level locomotion policy, which embodies the reflexive capabilities of "System 1." The planner performs deliberate pathfinding over a given environment, determining optimal paths from a

start to a goal, while the locomotion policy—trained for robust execution on diverse terrains—handles the execution of these paths, responding instinctively to local challenges.

One popular approach for quadruped navigation, as demonstrated in ANYMal Parkour [3], employs a fully-learned hierarchical framework. This method trains advanced locomotion skills tailored to specific obstacles, such as walking, climbing, and jumping, and uses a high-level policy to dynamically select and execute these skills across various terrains. By explicitly modeling the capabilities of each skill, the navigation policy adapts its behavior to suit the terrain and task requirements.

While this approach is undeniably powerful, it suffers from several limitations:

1. **Data Intensity:** Training a repertoire of specialized locomotion skills requires extensive data and computational resources, making the process resource-intensive and time-consuming.
2. **Increased Complexity:** The reliance on a navigation policy to dynamically control skill selection complicates the learning process, further increasing its data requirements.
3. **Lack of Long-Horizon Reasoning:** Fully-learned systems lack explicit planning mechanisms, making them less effective at reasoning over extended sequences of actions. This can lead to suboptimal performance in tasks requiring global optimization.
4. **Generalization Challenges:** Learned systems often struggle with out-of-distribution scenarios, where the skills or navigation policies fail to generalize effectively.

To address these limitations, our framework integrates a **generalist locomotion policy** with a **traditional planner**, adopting a hybrid approach that balances learning and planning:

- The **generalist locomotion policy** is trained to handle a broad range of obstacles without requiring task-specific training, reducing data demands and ensuring robust execution across diverse terrains.
- The **traditional planner** provides global reasoning over optimal paths, enabling long-horizon navigation and better generalization to unseen environments.

By combining these components, our framework ensures adaptability, efficiency,

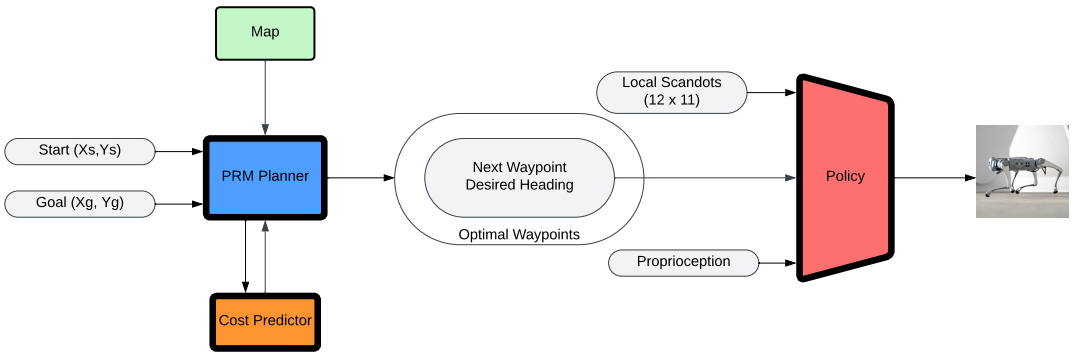


Figure 3.1: S3D-OWNS proposes a navigation framework for high-DOF systems like quadrupeds to leverage its skills to traverse a complex 3D world efficiently. This framework comprises of the three main modules, namely the Policy, the Cost Predictor and the PRM Planner.

and generalization, avoiding the pitfalls of fully-learned approaches while leveraging the strengths of learned locomotion.

The proposed system, S3D-OWNS, integrates three key components to achieve efficient quadruped navigation in complex environments:

1. A **generalist goal-conditioned locomotion policy**,
2. A **cost predictor** for reasoning over the optimality of paths,
3. A **multi-stage planner** for generating feasible and efficient paths.

In the following subsections, we provide a detailed overview of each component and their interactions, illustrating how they collectively address the challenges of long-horizon navigation. Our framework is designed to maintain optimality across key metrics, including energy efficiency, risk mitigation, and time minimization. By leveraging the quadruped's inherent agility, the system prioritizes effective obstacle traversal over simple avoidance, enabling efficient and robust navigation through complex terrains. Additionally, the planning process accounts for turning constraints, ensuring smooth, stable paths that minimize the risk of instability or mechanical strain caused by sharp turns.

3.2 Methodology: S3D-OWNS

3.2.1 Goal-Conditioned Generalist Policy

Our generalist locomotion policy builds upon the Extreme Parkour approach [1], introducing significant modifications to enhance versatility and robustness across a wider range of scenarios. Unlike the original policy, which was trained for specific parkour maneuvers with meticulously designed waypoints placed directly on obstacles, our policy is engineered to handle diverse edge cases and environments with "distractors." This robustness ensures seamless integration with the high-level planner, enabling reliable performance even in complex and unpredictable settings.

A key innovation of our approach lies in its goal-conditioning mechanism. In the Extreme Parkour policy, the oracle heading used during training is conditioned on the heading to the next waypoint as well as the heading to the waypoint after that, both computed with respect to the robot's current xy-position. In contrast, our locomotion policy retains this conditioning but introduces additional features: it also considers the distance to the next waypoint relative to the robot's current position and a target heading defined by the line joining the current waypoint and the next waypoint. This target heading specifies the robot's orientation upon reaching the next waypoint, providing a more precise goal for its trajectory. This design enables the planner to implicitly govern the robot's heading direction through the placement of waypoints, providing greater flexibility and precision in navigation. The details and implications of this mechanism are discussed further in this chapter.

We train the locomotion policy (figure 3.2) using model-free reinforcement learning (RL), specifically Proximal Policy Optimization (PPO) [10], in simulation. The policy takes as input the robot's proprioception, scandots, and next-goal information, which is encoded in the following components:

1. **Go-to Heading Direction:** The direction to the next waypoint relative to the robot's current position.
2. **Go-to Distance:** The distance between the robot's current position and the next waypoint.
3. **Target Heading Direction:** A heading defined by the direction to the next

waypoint relative to the current waypoint position.

4. **Subsequent Waypoint Direction:** The direction to the next-to-next waypoint relative to the robot's current position.

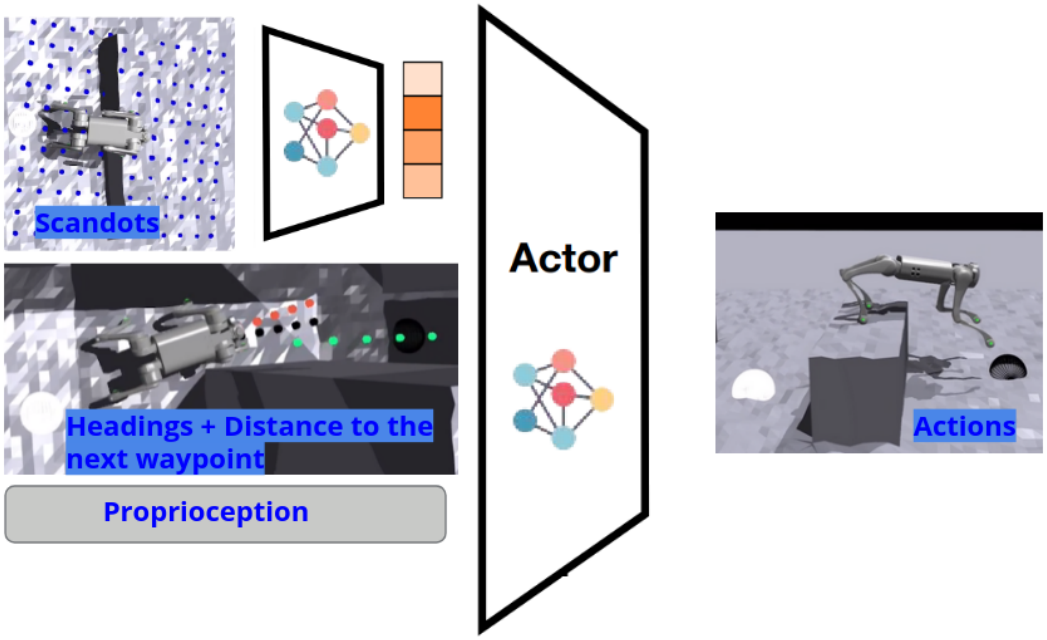


Figure 3.2: Reinforcement learning (RL) is used to train a locomotion policy with access to privileged information, including environment parameters and scandots, as well as task-relevant data such as the heading direction from waypoints, the robot's current position, and the distance to the next waypoint.

The policy is trained to perform diverse locomotion tasks, including walking, climbing up and down, jumping, and navigating around the edges of obstacles within its visual range. Effective edge navigation is essential for walls that the robot cannot climb, as well as obstacles such as blocks and gaps, where improper handling of edges could result in instability or failure.

We train the generalized goal-conditioned policy in multiple stages using Deep Reinforcement Learning (DRL). This staged approach allows us to progressively build complex behaviors [4], starting with fundamental skills and gradually incorporating more advanced capabilities [7].

The training process consists of four main stages:

Stage I: Locomotion with the major skills

In this initial stage, the locomotion policy is trained to learn core skills (figure 3.3), including running on flat terrain, jumping over gaps, and climbing up and down blocks, simultaneously. This setup ensures the policy is reactive to the environment, enabling it to handle diverse scenarios autonomously. While this approach limits the planner's ability to explicitly dictate skill execution, it reduces the planner's reasoning burden, as the policy itself navigates various obstacles effectively.

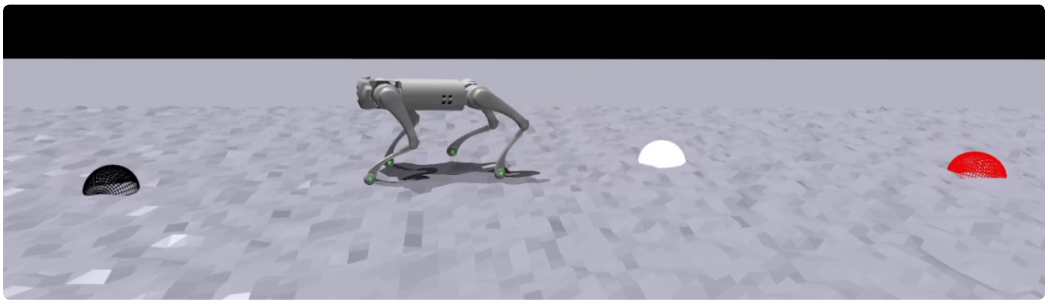
To train such a generalist policy, a diverse set of environments was created, allowing the policy to encounter a wide range of challenges. Using a curriculum learning approach, the training started with simpler environments and gradually introduced more difficult ones. This strategy improved sample efficiency and minimized the risk of catastrophic forgetting. In this stage, the policy was trained for 30,000 epochs on block, gap, and flat terrains, with the robot positioned to prioritize obstacle traversal over avoidance. Edge-specific training was deliberately excluded at this stage to focus on foundational skills.

The reward function follows a structure similar to the Extreme Parkour setup, combining task rewards with auxiliary rewards for safety, smoothness, stability, and posture style. Task rewards include velocity tracking and yaw tracking, where the tracking direction is determined by randomized waypoint placements on the terrain:

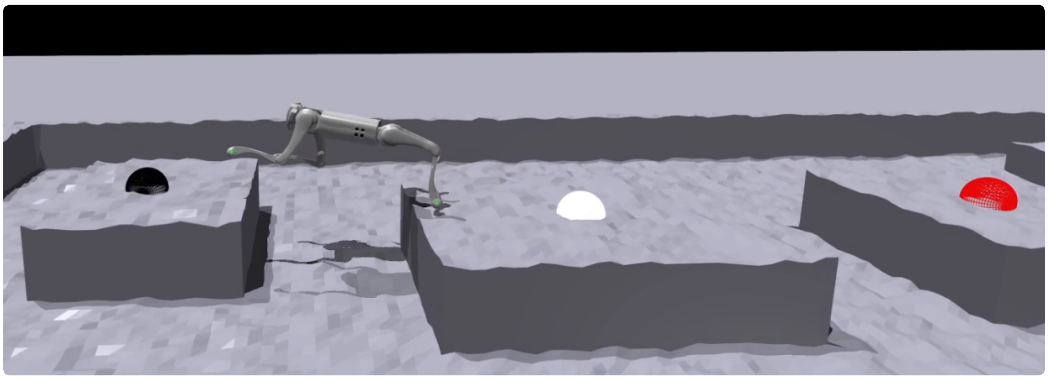
$$\hat{\mathbf{d}}_{\overrightarrow{\mathrm{RP}}} = \frac{\vec{\mathbf{p}} - \vec{\mathbf{r}}}{\|\vec{\mathbf{p}} - \vec{\mathbf{r}}\|} \tag{3.1}$$

Here, $\vec{\mathbf{p}}$ represents the next waypoint location, and $\vec{\mathbf{r}}$ is the robot's current position in the world frame.

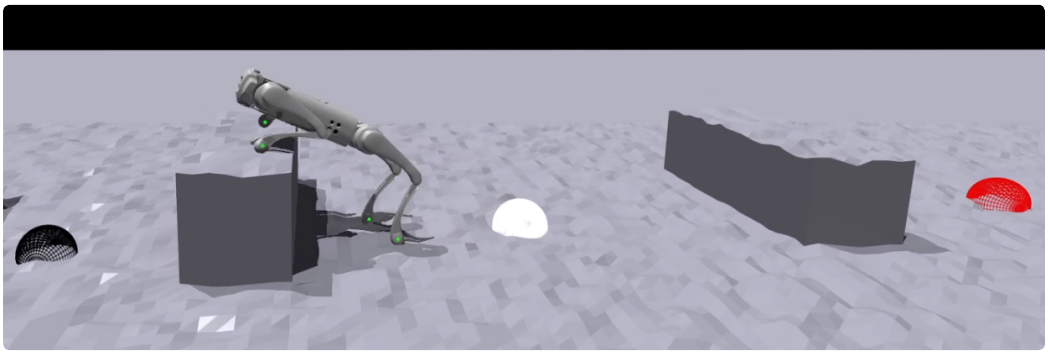
Training the policy using the same reward weights and PD gains for joint position-to-torque mapping as used by Cheng et al. [1] for the Unitree A1 robot did not directly translate to the Unitree Go1. Despite their structural similarities, the A1 robot has stronger motors compared to the Go1, leading to suboptimal performance across all skills during training. Without tuning the PD gains, the Go1 exhibited improper locomotion, such as moving on three legs while dragging one of its hind



Running



Jumping



Climbing

Figure 3.3: The major locomotion skills that the generalist policy was capable of.

legs.

Among the skills, jumping proved to be the most challenging to learn. To improve

performance, we modified the reward structure for this task by removing penalties for vertical (z-axis) velocity. However, when the z-velocity penalty was entirely removed, the robot developed an undesirable hopping behavior. Striking the right balance in the reward design was crucial for enabling stable and effective jumping.

Stage II: Edge Handling and Sideways Motion Penalty

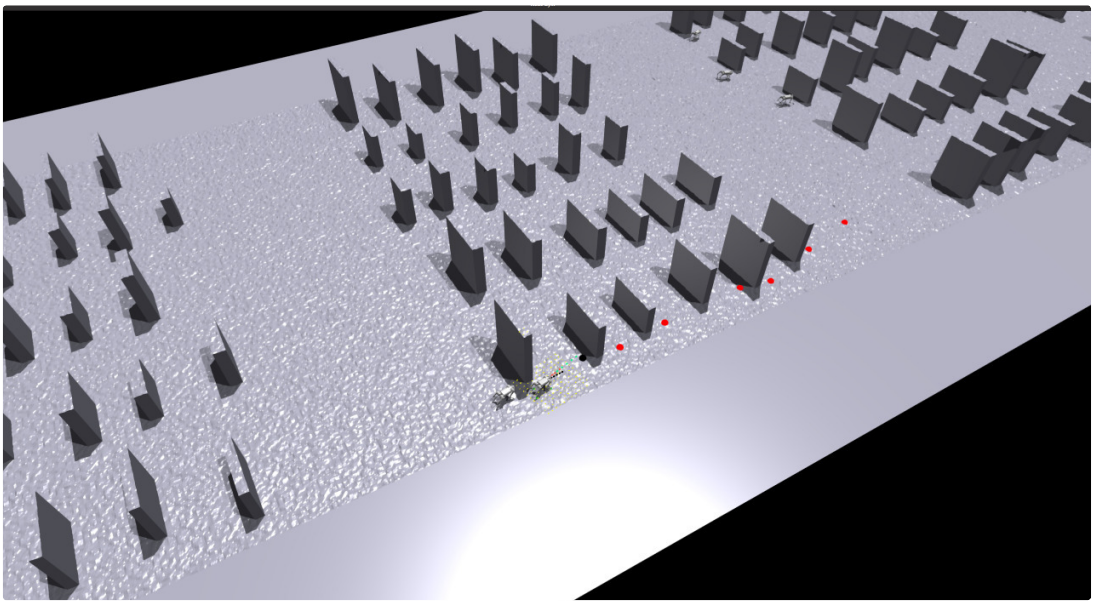


Figure 3.4: Training the locomotion policy to go around wall edges.

Building on the core skills trained in Stage I, we observed that the policy struggled significantly in scenarios where waypoints were placed near the edges of gaps and blocks. Performance degradation often resulted in the robot tripping, leading to instability. For walls, the robot exhibited a tendency to get stuck, as it lacked the ability to navigate around them. Instead, it would continue heading straight toward the next waypoint, even when a wall obstructed its path.

To address these limitations and to ensure that the planner need not filter waypoints based on their proximity to obstacle edges, we trained the policy specifically for edge scenarios. This enhancement simplifies the planning process, as it eliminates the need for reasoning about waypoint placement near obstacles.

To achieve this, we introduced an additional 3,000 epochs of training with the robot intentionally positioned near the edges of various obstacles. We increased stumbling penalties to promote safer navigation. Stumbling was defined as a specific type of collision where the horizontal component of the contact force exceeds the vertical component by a significant margin. Furthermore, we implemented a sideways motion penalty, recognizing the limitations imposed by the front-facing depth camera.

Terrain-specific adjustments were also introduced. For example, yaw tracking rewards were reduced by 50% in these edge-focused environments, granting the robot greater freedom to modify its orientation to navigate around obstacles effectively. This flexibility proved critical for traversing complex edge scenarios without compromising the robot's stability.

With these modifications, the policy became more adept at edge traversal, reducing the risk of instability and improving its ability to navigate around walls and other obstacles autonomously. These enhancements mark a critical step in enabling robust and versatile locomotion, bridging the gap between high-level waypoint generation and effective low-level execution.

Stage III: Distractor Robustness

To develop a locomotion policy that generalizes beyond specific training setups, it is essential to account for real-world scenarios where irrelevant or unexpected obstacles may appear. While traditional locomotion training often aligns closely with testing setups to benchmark performance in controlled environments, achieving robustness in navigation demands that the policy can distinguish between task-relevant obstacles and unrelated environmental noise. Similar to approaches in computer vision where distractors are introduced to improve noise resilience, we integrated distractors to enhance the policy's ability to stay focused on the task.

In this stage, we trained the policy to handle distractors such as additional gaps, blocks, and walls not directly relevant to the navigation task but present within the robot's visual range (see figure 3.5). By incorporating these elements into the environment, we ensured the policy would avoid unnatural or reactive behavior caused by irrelevant objects. This phase of training, spanning 3,000 epochs, emphasized robust navigation by teaching the robot to prioritize task-relevant information while

ignoring non-essential details.

The inclusion of distractors significantly improved the policy's ability to handle complex environments where irrelevant elements might otherwise cause confusion or instability. This enhancement ensures that the robot's focus remains on its primary navigation objectives, paving the way for more resilient and adaptable performance in real-world scenarios.

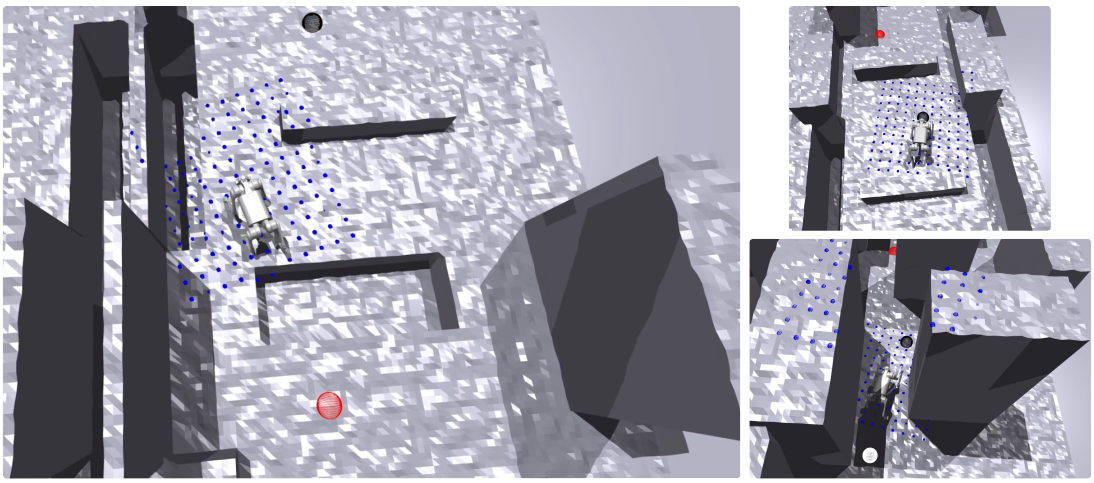


Figure 3.5: Left: Policy trained to handle overcoming block and gap edges with various distractors around them. Top Right: Policy trained to go across blocks and gaps with various distractors on the sides. Bottom Right: Policy trained to go around walls with other walls as distractors on the sides.

Stage IV: Heading Control

In this final stage, we aimed to enable the planner to control the robot's heading without the need to explicitly plan over an additional state, such as the robot's yaw. By avoiding explicit yaw planning, we reduce computational complexity, thereby mitigating the curse of dimensionality where planner computation time increases with additional state variables.

To achieve this, we designed the policy such that at any given waypoint, the planner inherently knows the robot's heading. Specifically, the policy was trained to orient the robot's heading direction along the line connecting the previous waypoint

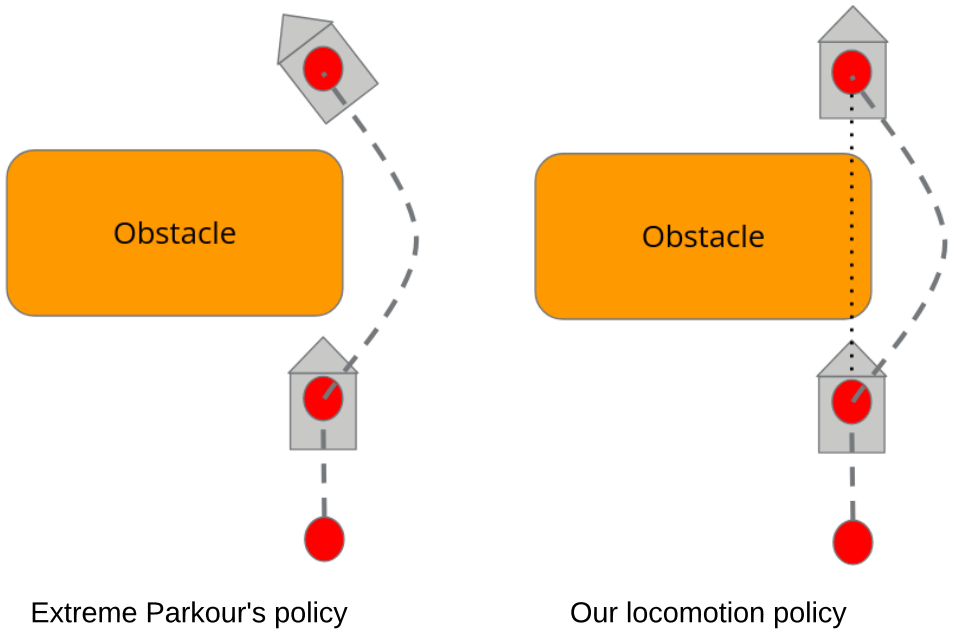


Figure 3.6: Difference in heading due to modification in the goal-conditioned policy.

to the current waypoint (see figure 3.6). This eliminates the need for the planner to explicitly compute yaw trajectories while ensuring consistent alignment with the intended heading direction.

In this stage, the policy underwent an additional 3,000 epochs of training to refine its ability to align the robot's yaw with the desired target heading. The heading direction, $\hat{\mathbf{d}}_{\overrightarrow{\mathrm{P_{curr}P}}}$, is mathematically defined as:

$$\hat{\mathbf{d}}_{\overrightarrow{\mathrm{P_{curr}P}}} = \frac{\vec{\mathbf{p}} - \vec{\mathbf{p}_{\mathrm{curr}}}}{\|\vec{\mathbf{p}} - \vec{\mathbf{p}_{\mathrm{curr}}}\|} \tag{3.2}$$

where $\vec{\mathbf{p}}$ represents the location of the next waypoint, and $\vec{\mathbf{p}_{\mathrm{curr}}}$ is the location of the current waypoint in the world frame.

Rather than explicitly rewarding yaw alignment, we incorporated heading control into the task completion criteria. The robot is required to realign its yaw within $\pm 5°$

of the desired heading before it can proceed to the next waypoint. This approach encourages the robot to naturally adjust its heading as part of achieving its navigation goals, promoting smooth transitions and maintaining planner simplicity.

Additionally, this approach allows the planner to generate paths that limit abrupt changes in the robot's heading direction. By avoiding sharp heading transitions, the robot remains within safe operational limits, reducing the risk of slipping or instability during movement. This ensures smoother and safer navigation across challenging terrains.

Throughout all stages, we leveraged transfer learning techniques [16] to efficiently build upon previously learned skills, significantly reducing the sample requirements for subsequent training stages.

The resulting goal-conditioned policy showcases remarkable adaptability, enabling the robot to navigate diverse terrains, handle edge cases, and maintain smooth and stable headings. By allowing the planner to control the robot's position explicitly through waypoints while implicitly directing its heading along the path, this approach ensures seamless integration of spatial and directional constraints. The implicit heading control not only simplifies the planner's computational burden but also avoids abrupt turns that could lead to instability or excessive joint torques, ensuring safer and more reliable navigation.

This combination of adaptability and implicit heading control empowers the planner to generate efficient, sophisticated trajectories that optimize both safety and performance, paving the way for robust navigation in complex and dynamic environments.

3.2.2 Cost Predictor

Cost predictors are a fundamental component of the S3D-OWNS framework, inspired by [2]. Their primary role is to estimate the cost of traversing between waypoints, enabling the planner to calculate optimal paths from start to goal. We experimented with three types of cost predictors: *energy cost*, *time cost*, and *success cost*, each serving a unique purpose:

- **Energy Cost**: Predicts the energy expenditure (in Joules) required for traversal.
- **Time Cost**: Predicts the time (in seconds) needed to reach the next waypoint.

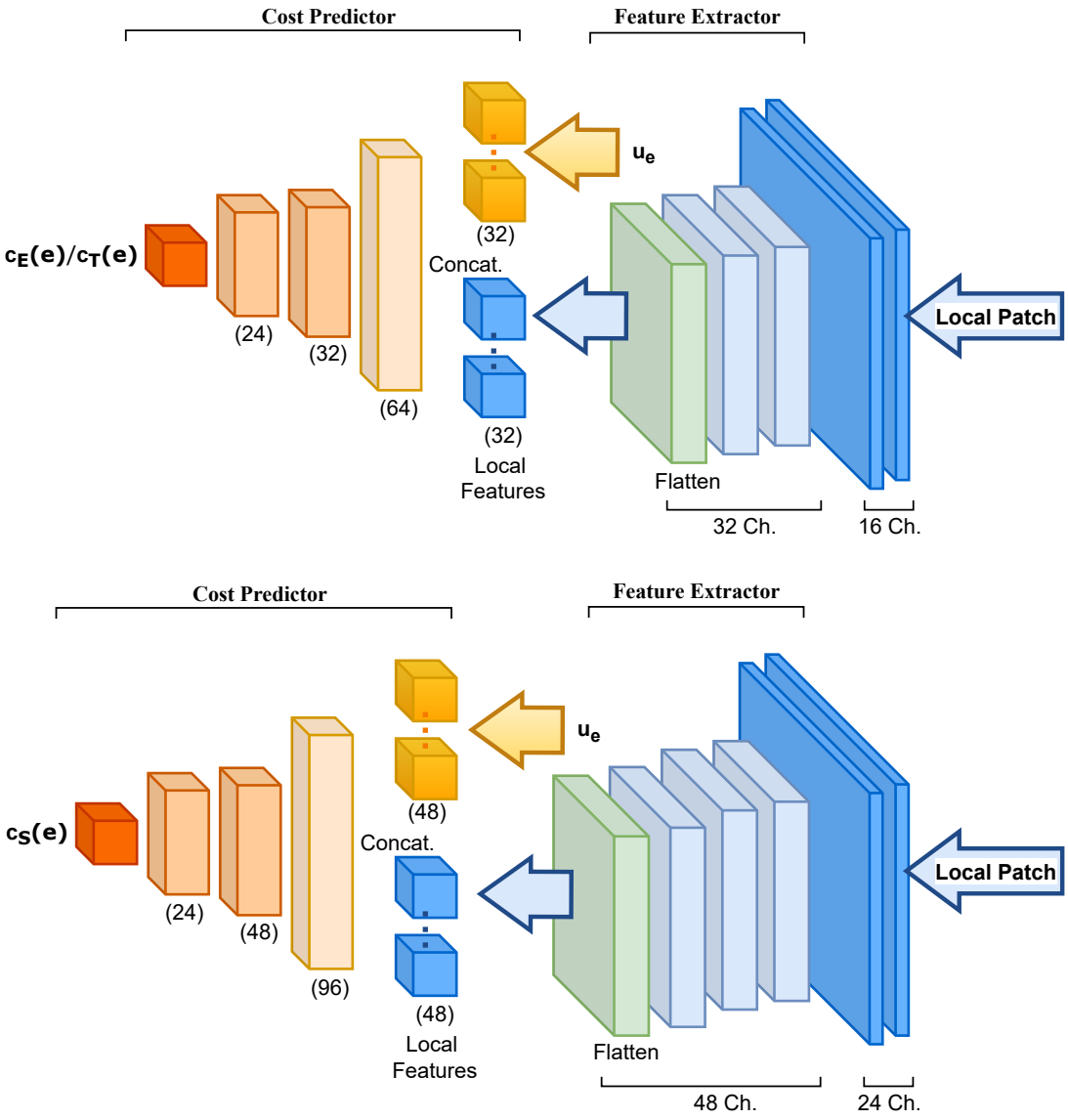


Figure 3.7: Cost Predictors: In the top, the cost predictor architecture for energy cost $c_E(e)$ or time cost $c_T(e)$. In the bottom, the cost predictor architecture for the success $c_S(e)$ classifier.

- **Success Cost**: Predicts whether the robot is likely to successfully traverse a path segment, given the terrain and heading requirements.

Success Predictor

The success predictor is formulated as a binary classification problem, where success is defined as a probability $\geq 50\%$. This threshold ensures robustness by focusing on conditions where the robot is more likely than not to succeed. Initially, the network was trained to predict the success rate directly as a regression problem, but it struggled to learn this complex mapping due to the difficulty of estimating precise fractional success rates (e.g., predicting 4 out of 10 robots succeeding). Simplifying the task to classification improved both learning efficiency and prediction accuracy.

Network Design

The cost predictors share a unified architecture that combines convolutional and fully connected layers (see Figure 3.7). Inputs to the networks include:

- **Edge Features (u_e)**: Comprises essential geometric metrics that facilitate accurate cost estimation. These include:
  - **Euclidean distance (Δd):** The straight-line distance between consecutive waypoints, representing the spatial separation that the robot must traverse.
  - **Change in heading angle (δ):** The angular difference between the current heading direction (defined by the line joining the previous waypoint to the current waypoint) and the desired heading direction (defined by the line joining the current waypoint to the next waypoint). This captures the sharpness of the turn required for traversal, which is critical for understanding energy and time costs as well as stability considerations.
  - **Derived terms ($\sin\delta$ and $\cos\delta$):** These trigonometric components of the heading angle change provide a more granular and interpretable representation of the angular difference. By decomposing the angle into its sine and cosine, the cost predictors can leverage these continuous and smooth features to better capture directional nuances. This is particularly useful for understanding the extent of the turn (e.g., left or right) and its impact on traversal cost. Additionally, these terms allow the model to process angular relationships more effectively in scenarios involving sharp turns or complex trajectories.

These features encapsulate the geometric relationship between waypoints, providing the foundation for assessing traversal feasibility and associated costs.

- **Local Terrain Patch**: Encodes height map information around the robot, offering rich contextual details about the surrounding terrain (Figure 3.8). The local patch used for cost prediction was designed to be significantly larger than the scandot input size used by the policy. Specifically, the scandot size was set to 1.65 m along the robot's length and 1.5 m along its width, while the local patch covered an area approximately five times the robot's length (5×0.688 m) along its length and four times the robot's width (4×0.688 m). This larger local patch size ensures that the next waypoint is neither too close, which could cause the robot to slow down unnecessarily, nor too far, which might require the cost predictor to reason about multiple obstacle types simultaneously—thereby increasing the risk of prediction errors. Like the scandots, the local patch heightfield is offset towards the front of the robot, prioritizing more points in front of the robot to better anticipate upcoming obstacles and terrain features.

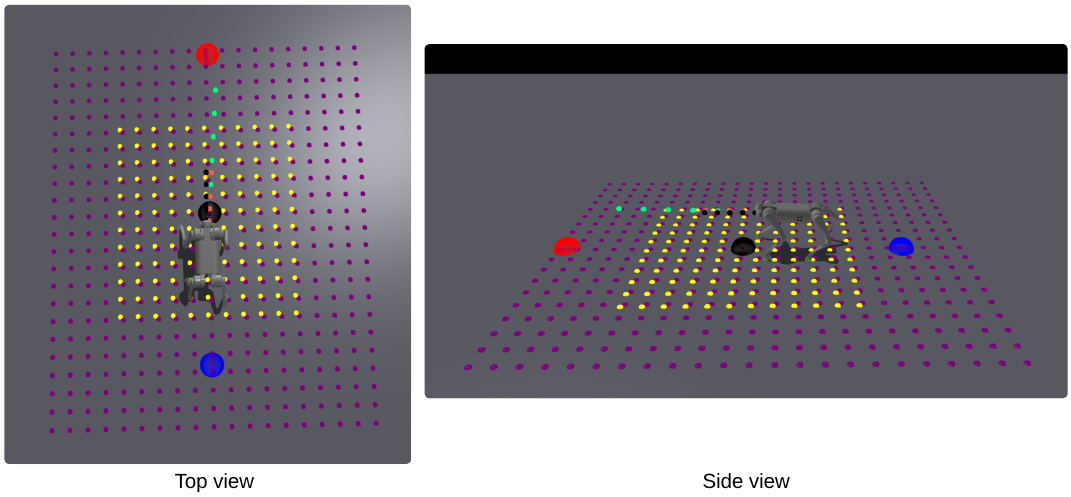


Figure 3.8: Local patch: The locomotion policy is trained on a smaller patch (scandots shown in yellow) as compared to the local patch (shown in purple) used by the planner. The size of the scandots used in the policy is 1.65m x 1.5m and the size of the local patch is 3.44m x 2.752m.

The architecture integrates a CNN backbone to process the local terrain patch and fully connected (FC) layers to handle edge features. These two representations are concatenated, forming a unified feature vector, which is passed through additional non-linear FC layers to generate the final predictions. The outputs of the network are as follows:

- **Energy Cost ($c_E(e)$):** Predicts the energy expenditure (in Joules) required for the robot to traverse the edge.
- **Time Cost ($c_T(e)$):** Estimates the time duration (in seconds) needed to move from one waypoint to the next.
- **Success Probability ($c_S(e)$):** Provides a binary classification to determine if the probability of successfully reaching the next waypoint exceeds 50%. This prediction aids in identifying traversable paths while accounting for obstacles and the robot's heading alignment.

Data Collection and Training

The datasets for training these predictors were collected in simulation, ensuring a diverse and balanced representation of terrain types and obstacle configurations. Energy and time predictors were trained as regression models, leveraging ground truth from simulation experiments. The success predictor's labels were generated by simulating 10 parallel trials for each scenario and determining the proportion of successful traversals.

By integrating these predictors into the planning framework, the robot is equipped with the ability to assess traversal feasibility, prioritize paths with minimal energy and time costs, and dynamically adapt to changing terrain. This capability significantly enhances the planner's efficiency and robustness in complex, simulated environments.

3.2.3 Planner

Modern learning-based methods often struggle with decision-making in long-horizon tasks due to challenges in scalability and optimality. While advancements in Large Language Models (LLMs) show promise as potential planners, they currently cannot guarantee the level of optimality achievable by traditional planning methods. To

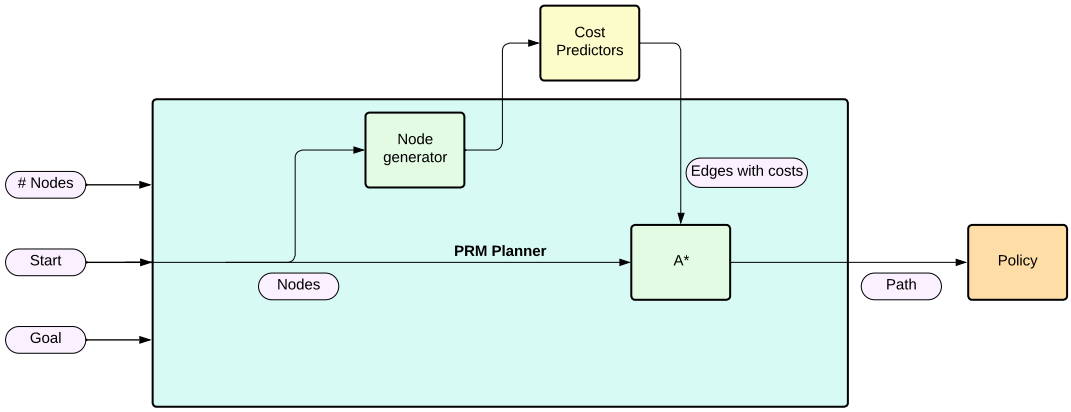


Figure 3.9: Flowchart of the PRM-based planning process, illustrating key components: node generation, cost prediction using energy, time, and success estimations, and path planning with A* search. The process begins with a defined start and goal, and the planner outputs an optimized path based on the specified task objective.

address these challenges, our framework leverages the strengths of a sampling-based planner as a global planner to generate safe, optimal paths for navigation. We define our planning configuration space

$$Q = \{x, y, \psi\}$$

and the subspace

$$q_{2D} = \{x, y\}$$

Where x and y are cartesian coordinates of the environment and ψ is the absolute heading of the robot. Our planner consists of three key stages:

1. **PRM Sampling in 2D Space:** The planning process begins by sampling waypoints in a 2D configuration subspace q_{2D} producing a node map for x and y coordinates. The number of samples, ranging from 700 to 1400 depending on the map size and complexity, is provided as input to ensure comprehensive coverage of the environment. This sampling ensures a well-distributed roadmap that adapts to varying environmental constraints and system limitations.
2. **Extra Node Generation for Heading:** To incorporate heading information

into the roadmap, we generate additional nodes representing the robot's orientation. This extension transforms the problem from a purely 2D task into a 3D planning problem with configuration space Q, embedding the heading as a critical dimension alongside the x and y coordinates. This integration allows the planner to account for both spatial and directional constraints, ensuring the planned paths align with the robot's physical capabilities and task requirements.

3. **A* Planner with Cost Predictor Integration:** The A* algorithm is utilized to search through the configuration space Q, incorporating heading information implicitly via the parent node's location. For each local action $a_i = \{\Delta d, \delta\}$, the planner queries the cost predictors to obtain edge costs based on the change in euclidean distance Δd and change in ψ, ie. δ mentioned in 3.2.2, including energy cost $C_E(e)$, time cost $C_T(e)$, and success probability $C_S(e)$. To mitigate the computational overhead caused by frequent GPU-to-CPU data transfers during cost queries, all edges, including those with heading dimensions, are pre-generated. GPU parallelization is employed to compute costs efficiently in batch.

The task objective, as defined by the user, can be one of the following: minimum energy, minimum time, or maximum success. Additionally, it can be a combination of success followed by minimum time or energy, enabling the generation of safer yet optimal paths. Figure 3.9 illustrates this process. Since the robot's heading is controlled by the provided waypoints (as discussed in Section 3.2.1), the waypoints cannot be sampled randomly, as would typically occur in a standard PRM formulation.

The heuristic for the A* planner is designed using a 2D backward Dijkstra search algorithm. This heuristic is informed by analytical cost approximations, which are derived from sparse data points collected through simulation experiments and subsequently curve-fitted for accuracy. Additionally, to avoid excessively risky paths, the success rate $C_S(e)$ is used as a cutoff criterion for both minimum energy and minimum time objectives. Specifically, edges with $C_S(e) < 0.5$ could be excluded from the search space, ensuring that the planner prioritizes feasible and reliable trajectories.

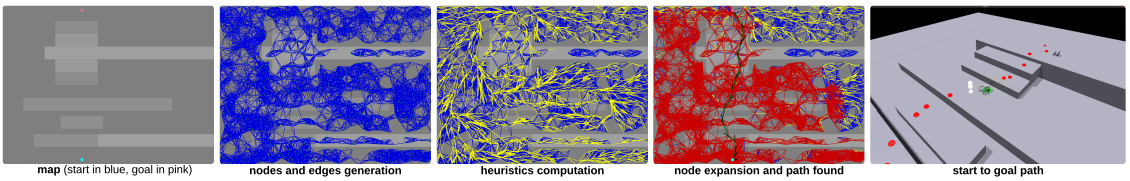


Figure 3.10: Stages of path planning and waypoint generation. The planner begins with a 2D grayscale map, which is converted into a heightfield in IsaacGym based on pixel values. Green nodes represent sampled waypoints, and valid blue edges are determined based on changes in heading direction. Next, yellow edges illustrate the heuristic connections used for planning. The expanded nodes and their corresponding red edges are shown in the subsequent stage. Finally, the planned red waypoints are visualized in the 3D world.

Chapter 4

Experiments

4.0.1 Analytical Costs

Before creating the large datasets for training cost predictors, we conducted controlled experiments in simulation to analyze how various input variables—such as the distance between waypoints, change in heading direction, and local patch features like block height or gap width—affect traversal costs. In these experiments, we varied only one input variable at a time, isolating its influence on the costs. The resulting data provided valuable insights into the relationship between input features and costs, enabling us to fit high-order polynomials to model these dependencies. These fitted polynomials represent the analytical costs and were subsequently used for heuristic computation in the backward-Dijkstra algorithm and as edge costs in analytical-cost-based planners. This analysis also served as a benchmark for evaluating the performance of predictor-based planners.

The experimental setup, as illustrated in Figure 4.1, demonstrates that the data collection process heavily relies on the placement of three consecutive waypoints, which define the turning angle. The waypoints were restricted to lie within a range of [0.5,2.5 m]. This range was chosen to ensure that the distance between consecutive waypoints was neither too short—avoiding situations where it would be smaller than the robot's size—nor too large, where the next waypoint would fall outside the local patch used by the planner.

Figures 4.2 and 4.3 summarize the results of multiple experiments, each performed

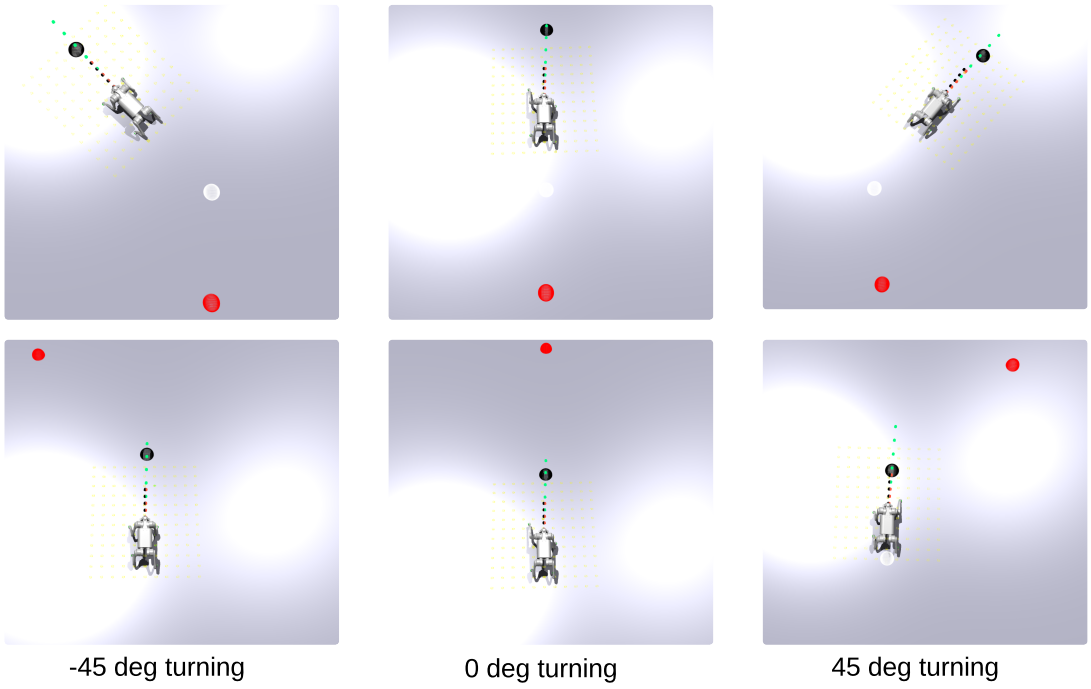


Figure 4.1: Analytical cost generation process for turning angles. 3-waypoints placement for the all data collection process. The turning angle is defined by the change in angle between the line connecting the first and second waypoints and the line connecting the second and third waypoints.

10 times to compute average costs for walking, climbing, and jumping. The results reveal that traversal costs are influenced by several factors, including the distance traveled, obstacle type, obstacle size, and turning angle required to reach the goal. To maintain consistency across experiments, we kept variables such as friction, smoothness, robot mass, and commanded velocities constant. Notably, the stochastic nature of the policy introduced variations across the runs, with more challenging tasks exhibiting higher variability. Only successful runs were considered when computing the average energy and time costs.

Key Observations

- **Energy and Time Costs for Walking:**
  - Both energy and time costs are linearly proportional to the distance traveled, as expected given the fixed commanded velocity.

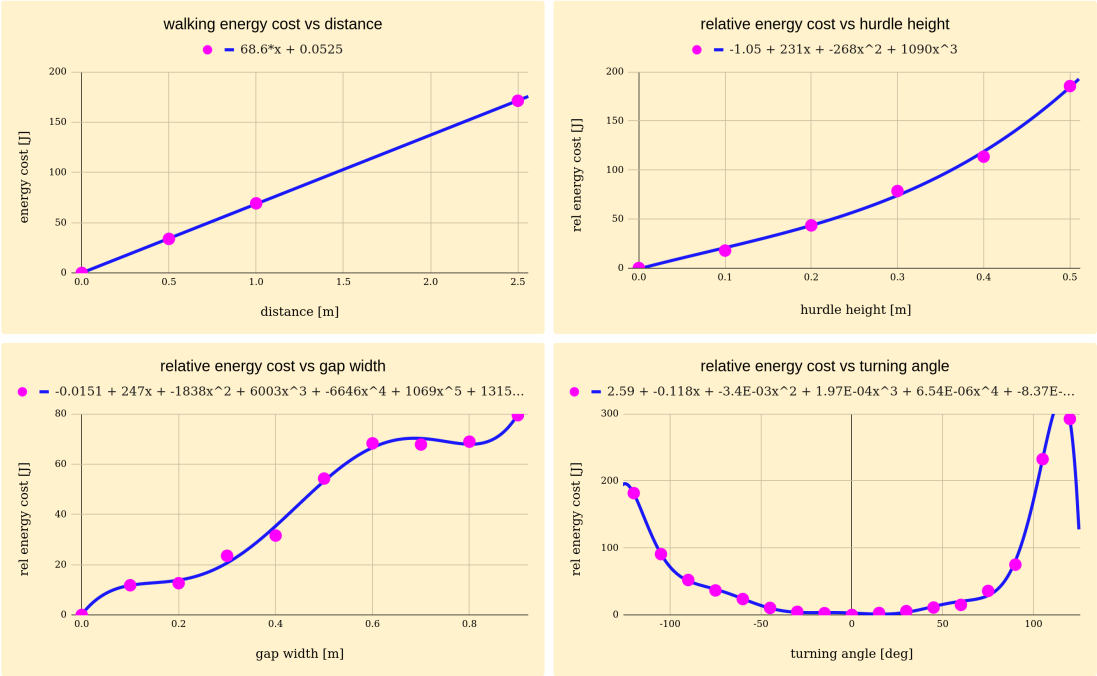


Figure 4.2: Analytical Energy Cost Functions.

- Energy costs exhibit additional increases when traversing hurdles that require climbing or jumping, with larger obstacles generally incurring higher costs.

- **Turning Costs:**
  - Turning costs depend on both the change in heading angle and the distance to the next waypoint.
  - Ideally, the cost plots for left and right turns should be symmetrical, though minor deviations were observed due to the robot's dynamics.
- **Time Costs for Obstacles:**
  - Time costs are generally lower for jumping over gaps and, in some cases, for climbing over obstacles.
  - Interestingly, there exists an optimal obstacle height for the robot, where climbing is both faster and more comfortable compared to other heights.

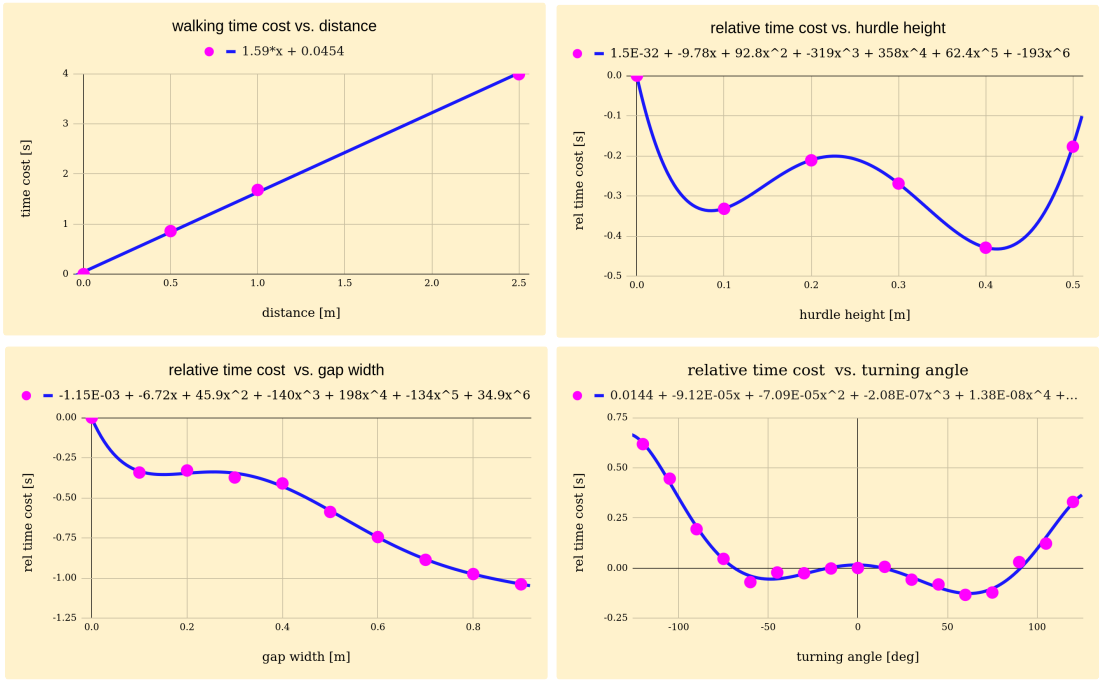


Figure 4.3: Analytical Time Cost Functions.

- Time cost plots for turning actions are approximately symmetrical, reflecting the additional time required for the turning maneuver.

These findings provide critical insights into how input features influence traversal costs, validating trends such as the proportionality of energy costs to distance and the symmetry of turning costs. These results highlight the importance of designing cost predictors that account for task-specific nuances while maintaining generality across different scenarios.

4.0.2 Dataset Collection

Datasets. To train the cost predictors effectively, we collected local motion costs by running the robot policy in simulation using Isaac Gym. We created three environment types for data collection: *block world*, *gap world*, and *hurdle world*, each featuring obstacles of varying sizes (Figure 4.4).

For *block* and *hurdle worlds*, obstacle heights ranged from 0.0 m to 1.0 m, with

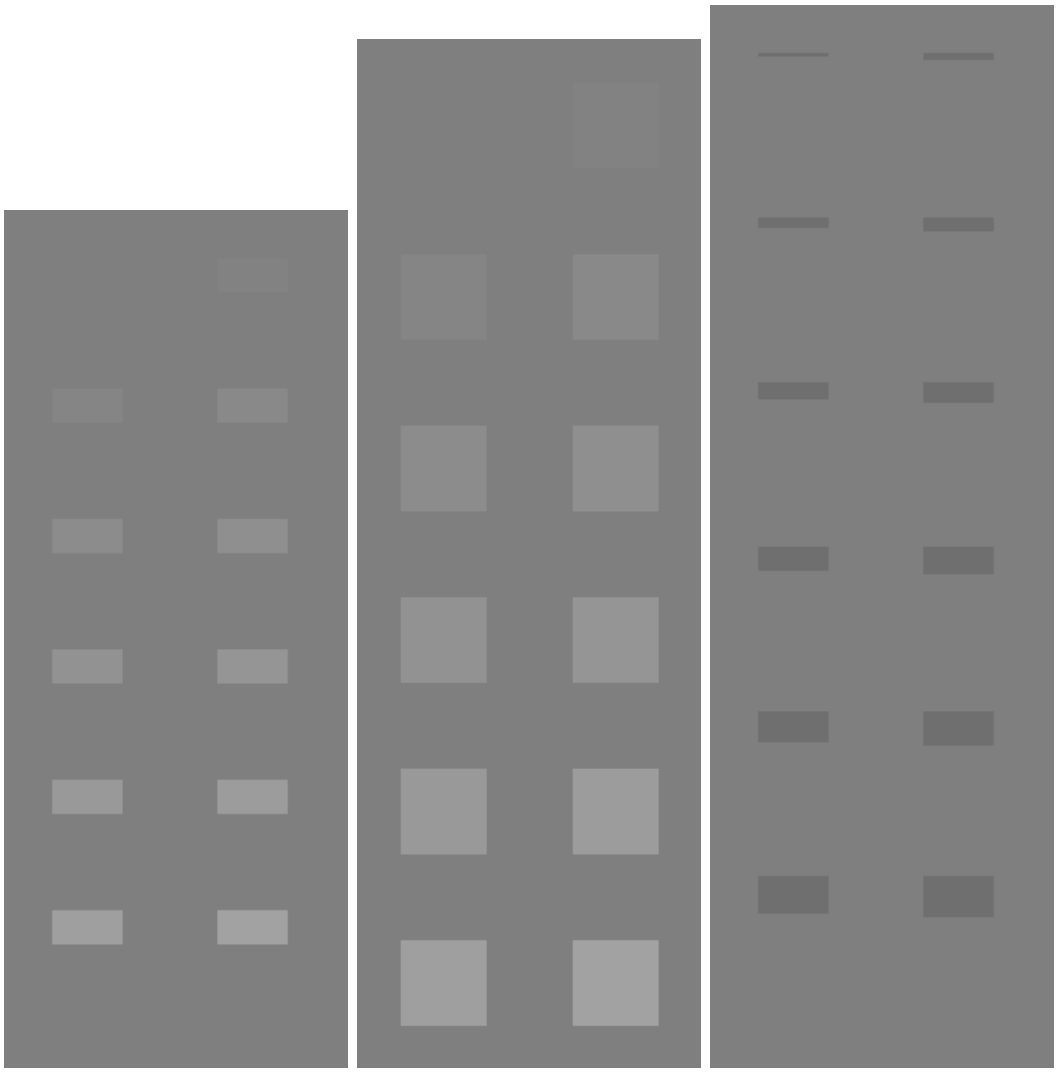


Figure 4.4: Hurdle World, Block World and Gap World. These are in the pixel space with varying sizes for the data collection process.

increments of 0.1 m. For *gap worlds*, gap lengths ranged from 0.1 m to 1.2 m, with the same increment. Based on the robot's policy, we observed that the robot could reliably climb obstacles up to 0.6 m and jump gaps up to 0.9 m. However, success depended heavily on the placement of the next waypoint, as this determines the angle at which the robot approaches the obstacle.

To expedite data collection, the simulation worlds were kept static throughout the process, significantly reducing environment initialization time. By avoiding frequent spawning of new environments—a time-consuming operation in Isaac Gym—we

reduced the total data collection time from 3-4 days to just one day.

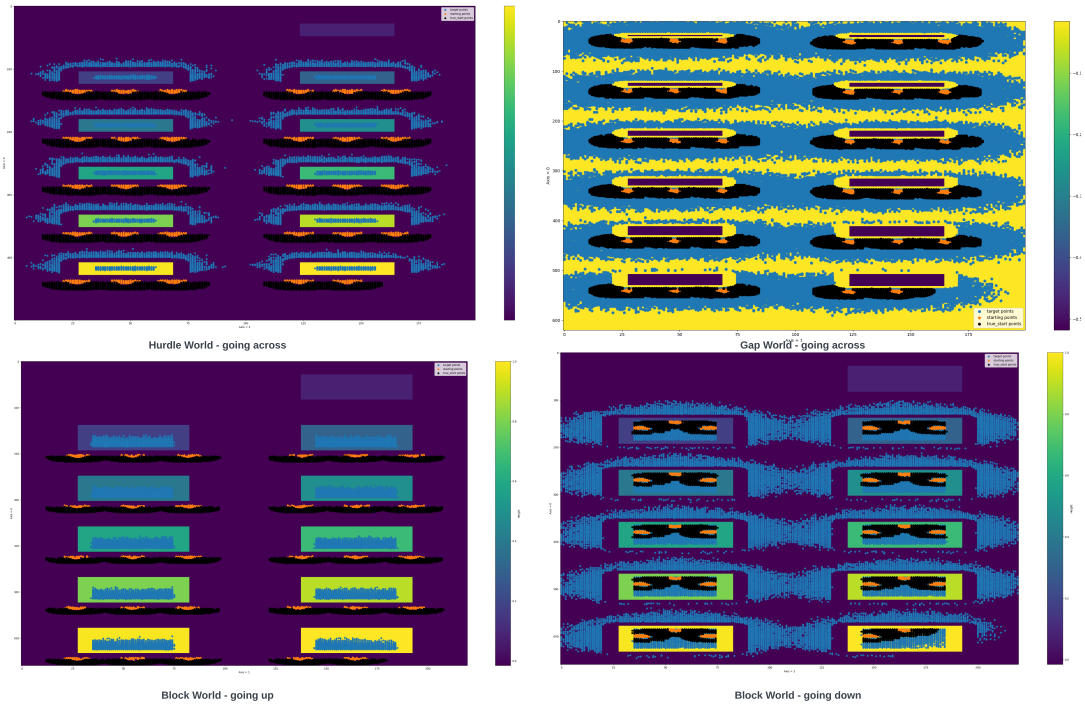


Figure 4.5: Automation of 3-wayponts generation process for different worlds. Reset points (in black), starting points (in orange) and goal points (in blue) for the all the worlds.

The robot was reset to a randomized location within a 0.5 m radius of a predefined starting point. For each starting point, 10 parallel agents collected data simultaneously, further accelerating the process. Before moving towards the target waypoint, the robot first aligned itself at the starting waypoint, ensuring a consistent initial heading. This added reset step introduced extra randomness to the robot's initial orientation by dropping it at slightly offset positions within the radius.

World and Waypoint Selection. For each task and environment type, valid starting and goal waypoints were selected based on feasibility constraints. These constraints included:

- **Obstacle Traversal:** In most cases, the goal must be on the other side of the obstacle. During the goal placement validity check, points within 0.3 m

(approximately half the robot's length) from the edge of obstacles are avoided. This constraint ensures that waypoints are not placed too close to obstacle edges, which could interfere with the robot's target orientation upon reaching the waypoint.

- **Turning Angle:** The angle required to traverse between waypoints must fall within a feasible range for the robot's capabilities. Waypoints resulting in turning angles greater than $\pm 120^\circ$ are discarded, as they exceed the robot's operational limits.
- **Local Patch Coverage:** The goal must lie within the local patch accessible from the starting waypoint, as illustrated in Figure 3.8. Points outside the local patch are excluded because the cost predictor network relies on local environment information within the patch. Without this data, it would be impossible to accurately predict the traversal costs due to incomplete environmental context.

To account for cases where the robot needs to adjust its final heading, padding was added around the edges of blocks and gaps. An automation script was used to generate approximately 3,000 valid waypoint pairs for each obstacle type, with the process completing in approximately 10 minutes.

Data Collection Metrics. During data collection, the following metrics were captured:

- **Energy Consumption:** The total energy consumption between the starting waypoint and goal waypoint was recorded. It is computed as $\sum(\boldsymbol{\tau} \cdot \boldsymbol{\omega})\Delta t$, where $\boldsymbol{\tau}$ and $\boldsymbol{\omega}$ are the vectors of joint torques and joint velocities, respectively, and Δt is the time step for each simulation frame.
- **Traversal Time:** The total traversal time from the starting waypoint to the goal waypoint was measured and computed as $\sum \Delta t$, where Δt represents the time per simulation step.
- **Success:** The success was determined based on whether the able to reach the goal or not before the end of the episode for each combination of reset position, starting point, and target point.

Each experiment was conducted with 10 robots spawned in parallel under identical initialization conditions, and the above metrics were recorded for each robot. Finally, the success rate was computed as $\frac{\text{Number of Successful Runs}}{10}$, representing the fraction of

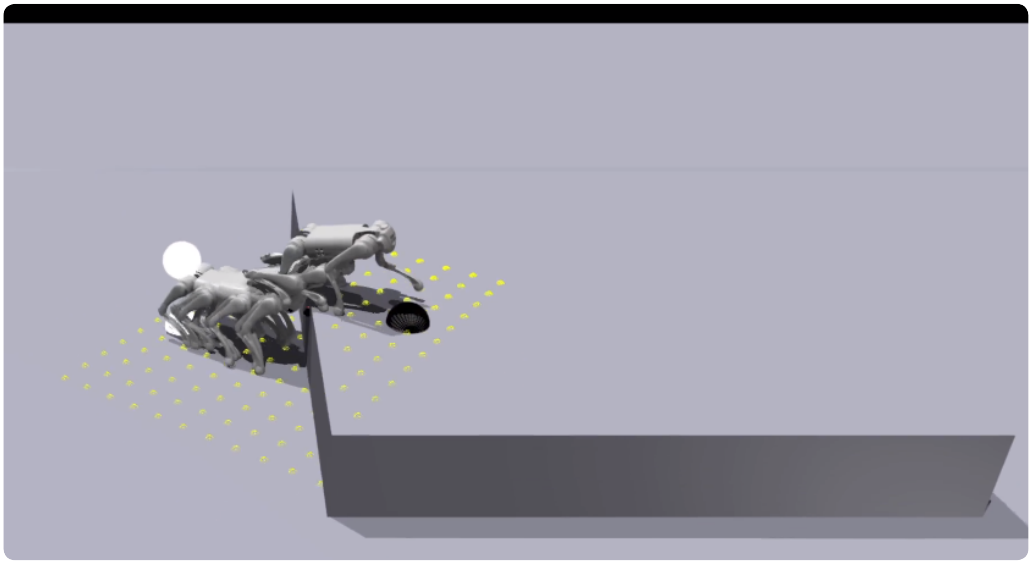


Figure 4.6: Data collection process: For each combination of reset position, starting point, and target point, 10 robots are spawned simultaneously in parallel, and relevant information for cost computation, such as energy consumption, traversal time, and success metrics, is logged.

robots that reached the goal. Average energy and traversal time were calculated using only the successful runs to ensure meaningful comparisons.

Utility of the Dataset. This comprehensive dataset serves as the foundation for training the cost predictors to estimate energy, time, and success costs associated with various navigation strategies. By providing accurate predictions, the cost predictors enable the planner to generate efficient, safe, and feasible paths, optimizing the robot's navigation performance.

Dataset Analysis. We analyzed the datasets to evaluate their distribution and identify trends before training the cost predictors.

As shown in Figure 4.7, we observed a significant data imbalance that could adversely affect the training process. To mitigate this, we included only data points with a success rate greater than or equal to 0.5 when training the energy and time cost predictors. This filtering reduced data imbalance and led to a dataset with less

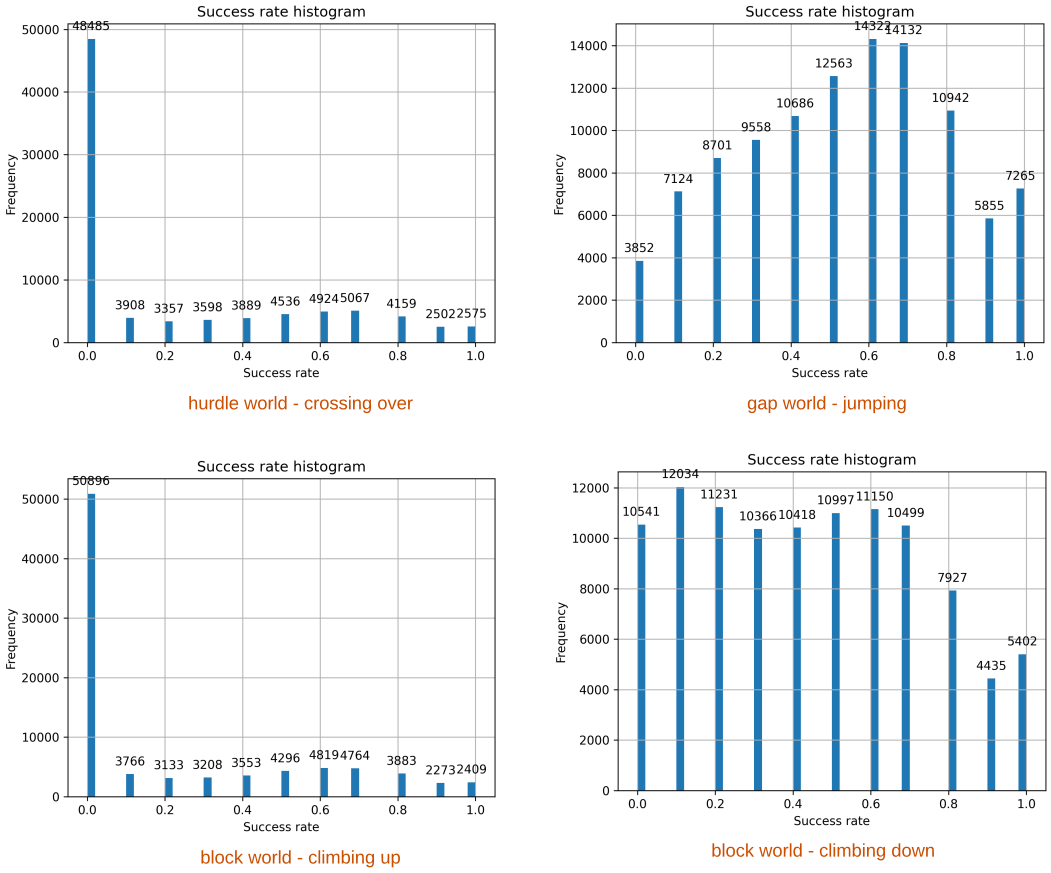


Figure 4.7: Success Rate vs Number of samples for all worlds.

variance, making it easier for the network to learn effectively.

For the success rate and success cost predictors, we discarded data points with zero success rate from the hurdle world (crossing over) and block world (climbing up) datasets. These data points correspond to obstacle heights greater than 0.6 m, which are not traversable by the robot and effectively act as walls. Since our planner already discards waypoints placed on such walls, these points were deemed irrelevant for training. Removing them prevents the success predictor from being biased by untraversable scenarios, avoiding pessimistic predictions that might incorrectly classify obstacles as insurmountable.

To prepare the data for training, we normalized the energy and time cost datasets

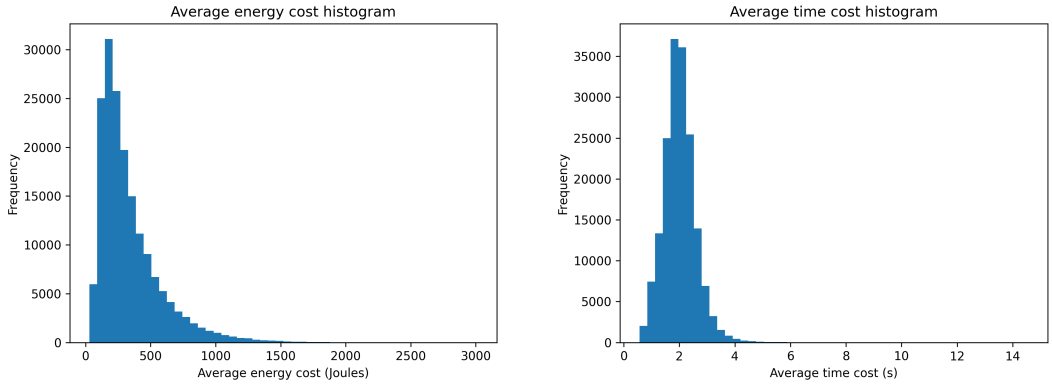


Figure 4.8: Distribution of energy costs and time costs for success rate greater than or equal to 0.5

using z-standardization, as these costs closely followed a normal distribution (Figure 4.8). For the success classifier, we applied min-max normalization. These normalization techniques ensured that the networks produced outputs within standardized ranges, facilitating more stable and effective learning.

To further understand the trends in the data, we plotted the distributions of energy (Figures A.1 to A.12) and time costs (Figures A.14 to A.25) separately for each environment and for different success rates. These plots illustrate the variations in cost values, represented using a color map, as functions of distance (x-axis), turning angles (y-axis), and obstacle heights (indicated by the size of the circular data points). The observed trends in these distributions align closely with those identified during the analytical cost computation process, thereby validating the datasets.

For the energy plots, it is observed that energy consumption increases with greater start-to-goal distances and sharper turning angles. Additionally, sharper turns are associated with lower success rates. Obstacles such as blocks, hurdles, and gaps further contribute to increased energy expenditure. Similarly, the time plots reveal that traversal time increases with longer start-to-goal distances and sharper turning angles. However, traversal time may decrease when jumping over gaps or climbing up and down blocks or hurdles, likely due to the fact that the policy has learned to increase speed to overcome such obstacles more successfully.

Examining the 3D plots (Figures A.13 and A.26), it becomes evident that the

time data exhibits a clearer trend with less noise compared to the energy data. This outcome is expected, as energy computation involves joint torques and joint velocities, which are inherently noisy. Consequently, it is more challenging for a neural network to accurately learn energy costs compared to time costs.

Cost Predictor Training Results. We trained the energy and time cost predictors using a balanced dataset comprising data points with a success rate of at least 0.5 and evaluated their performance by plotting the predicted costs against the ground truth costs (Figures 4.9, 4.10). For an ideal regressor, these plots would align along a straight line with a 45-degree slope. However, we observed that the energy cost predictor exhibited greater variance compared to the time cost predictor. This is expected since energy costs in simulation are inherently noisier than time costs, making the learning process more challenging for neural networks.

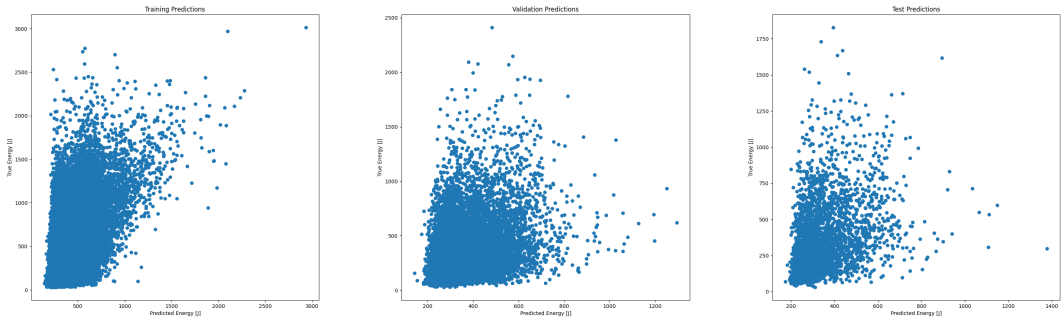


Figure 4.9: Groundtruth vs predicted costs for the energy cost predictor.

For the success classifier, we trained it using a balanced dataset with success rates ranging from 0.0 to 1.0. A success was defined as at least 5 out of 10 robots successfully navigating from the starting waypoint to the goal waypoint. The confusion matrix (Figure 4.11) shows that most points lie in the true positive or true negative categories. However, there are notable false positives and false negatives, which make the success classifier less reliable. This unreliability caused the planner to behave conservatively at times, failing to find feasible paths between the start and goal locations.

This issue is particularly pronounced in our sampling-based planner, where the number of valid points sampled across gaps or blocks is already limited. For a waypoint to be valid, it must lie within the local patch at the robot's current waypoint

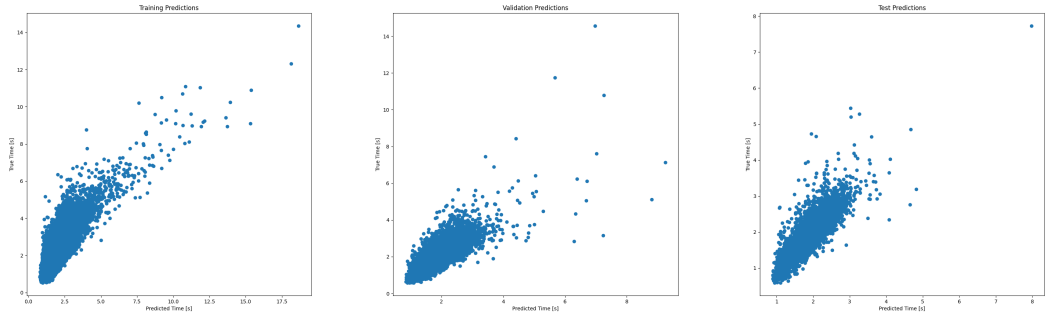


Figure 4.10: Groundtruth vs predicted costs for the time cost predictor.

and must also be at least half the robot's length away to avoid interfering with the target heading direction. Furthermore, at edges, if the robot approaches at angles greater than 30°, it struggles to overcome obstacles that are otherwise traversable. On test maps with gaps of 0.5–0.6 m or blocks 0.5 m high, if the success predictor incorrectly predicts false negatives for these sparse edges, the planner becomes unable to find a valid path. For this reason, we avoided using the success predictor when evaluating the planner's performance on our test maps.

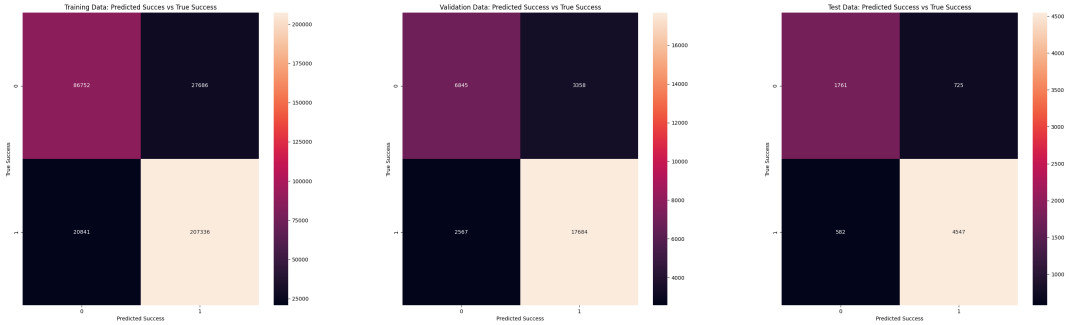


Figure 4.11: Confusion matrix for the success cost predictor.

Initially, we approached the success prediction problem as a multi-class classification task, aiming to predict the probability of success. However, the neural network often misclassified probabilities that were close to the ground truth. For instance, distinguishing between success rates of 0.2, 0.3, or 0.4 proved to be highly unreliable.

This challenge arises due to the following factors:

1. **Stochastic Nature of Policies:** The inherent randomness in the robot's policy introduces variability in outcomes even under similar conditions.
2. **Environmental Factors:** Critical factors such as friction and restitution coefficients, which significantly affect dynamics, were not included as inputs to the network.
3. **Task Complexity:** Agile maneuvers over challenging obstacles further increase uncertainty, making precise predictions more difficult.

To address these challenges, we simplified the problem to a binary classification task: determining whether the success rate for a given scenario exceeds 0.5. This simplification allowed the network to focus on broader success trends, resulting in a more reliable success predictor.

4.0.3 Planning Results

Experiments. Once we developed a generalist locomotion policy and trained cost predictors, the final step was to demonstrate how the S3D-OWNS framework, combined with our sampling-based planner, outperforms an obstacle-avoidance-based planner. The latter does not leverage the robot's capabilities and always opts for safe, obstacle-free paths. We conducted experiments on three different maps, each tailored to specific objectives:

- **Minimum energy:** Figure 4.12,
- **Minimum time:** Figure 4.13, and
- **Long-horizon minimum time:** Figure 4.14.

Minimum Energy Map

The $10\,\mathrm{m} \times 10\,\mathrm{m}$ map for minimum energy features two possible routes from the start to the goal location:

- **Route 1:** The robot climbs up and down a high block to traverse a shorter path.
- **Route 2:** The robot takes a longer path around a wall on flat ground.

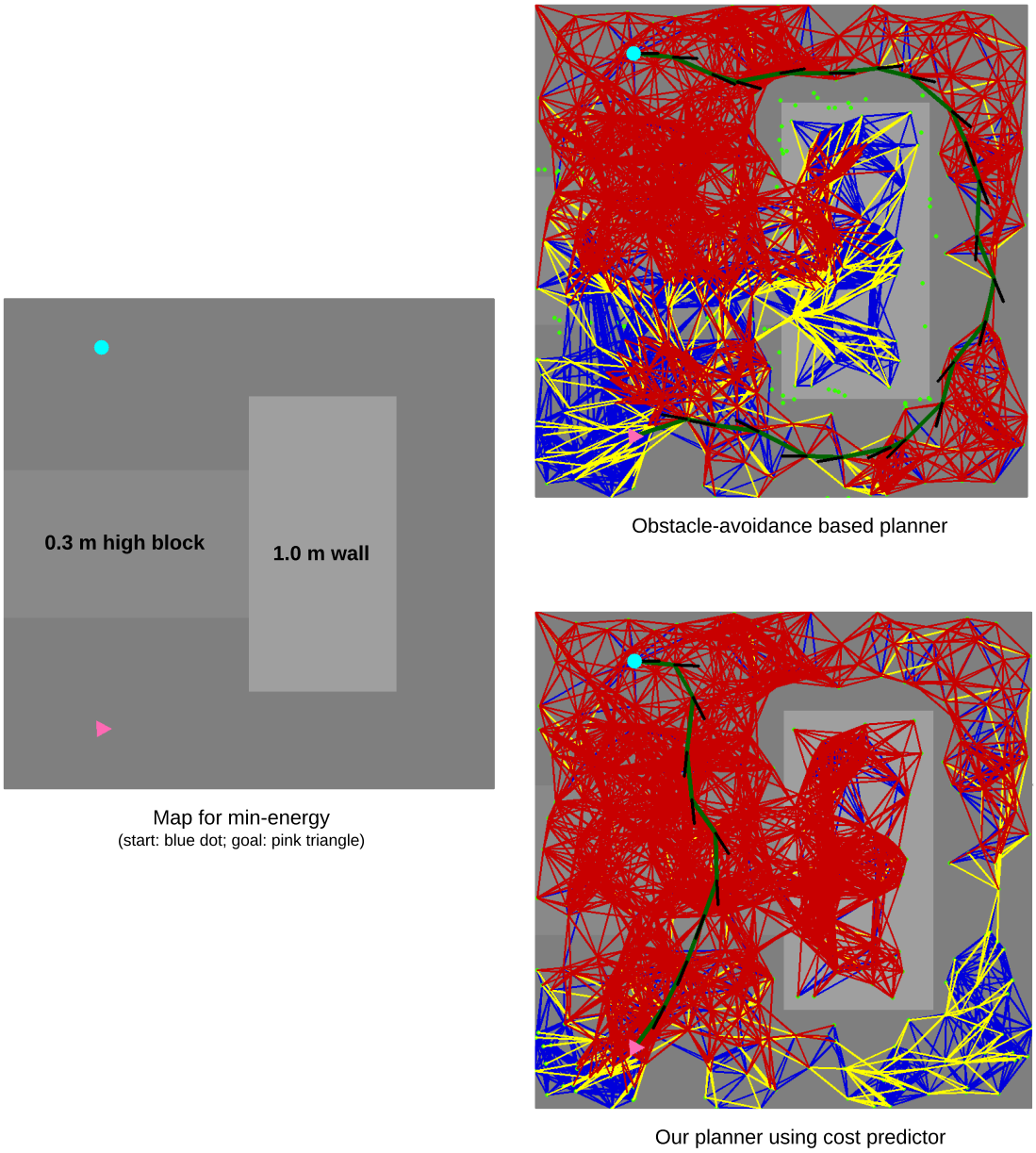


Figure 4.12: Obstacle-avoidance based planner vs our planner for minimum-energy path planning.

Although climbing requires extra energy, the significantly shorter distance of Route 1 results in lower overall energy consumption. Our planner, leveraging the learned energy cost predictor, selects the more energy-efficient path over the block.

This demonstrates how the planner optimally reasons about feasibility and cost, taking full advantage of the robot's climbing capabilities.

Minimum Time Map

Similar to the minimum energy map, the 10 m × 10 m map for minimum time also offers two routes from the start to the goal:

- **Route 1:** The robot jumps over two 0.6 m gaps.
- **Route 2:** The robot follows a longer, obstacle-free path with multiple turns and walls.

Jumping across the gaps is faster due to the robot's high speed during jumps, whereas the flat-ground path is slower because of the longer distance and additional turns. Our planner, using the learned time cost predictor, selects the faster path over the gaps, effectively utilizing the robot's jumping skills. In contrast, the obstacle-avoidance-based planner defaults to the longer, obstacle-free route.

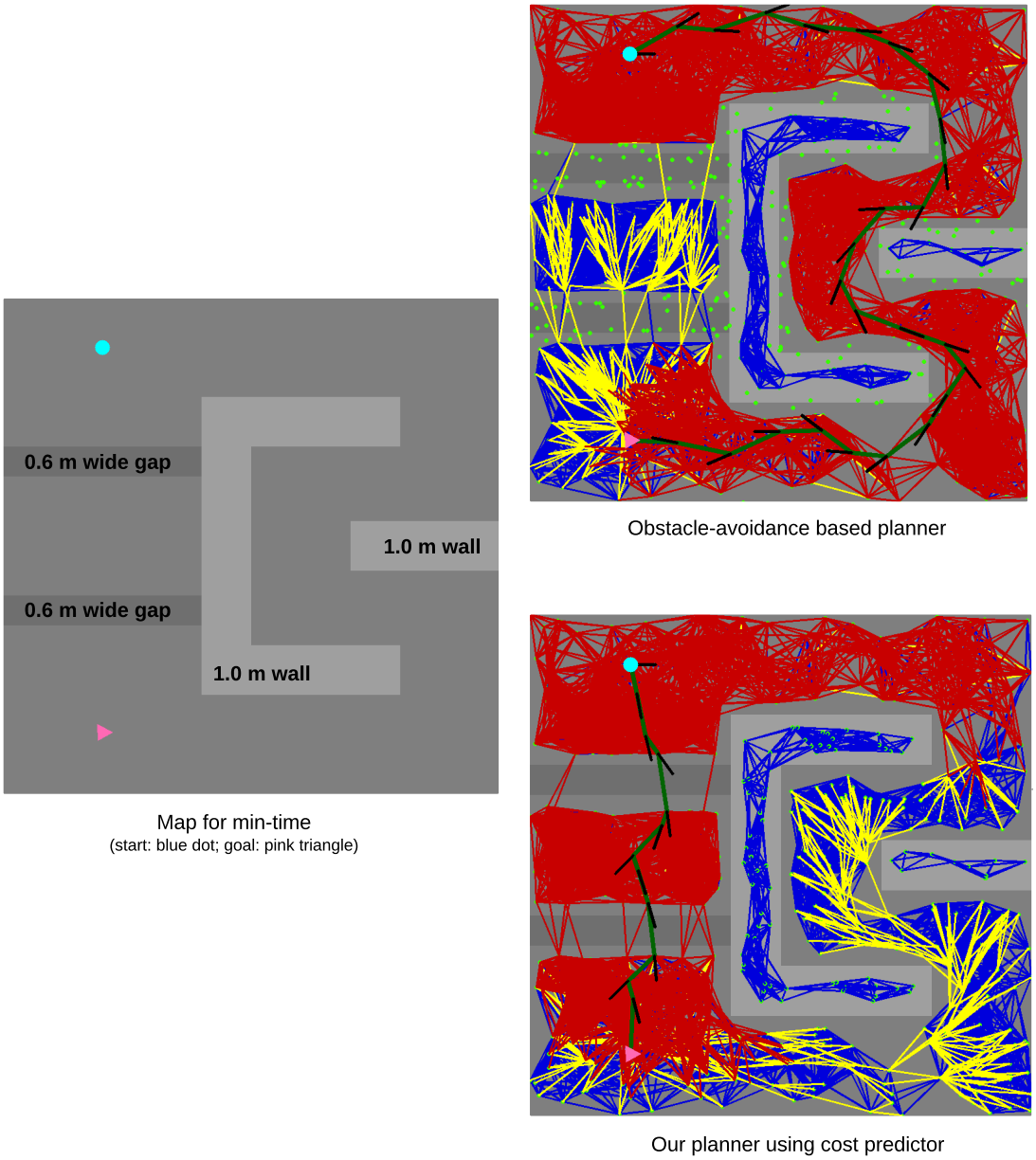


Figure 4.13: Obstacle-avoidance based planner vs our planner for minimum-time path planning..

Long-Horizon Minimum Time Map

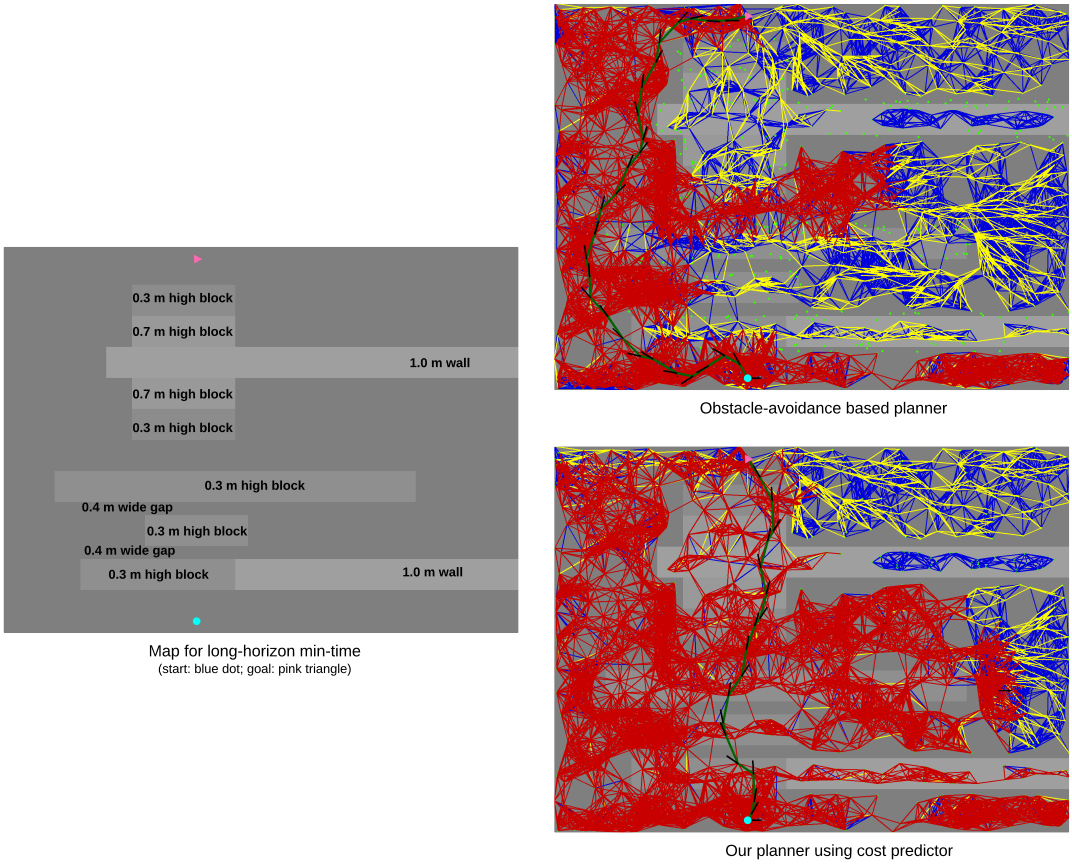


Figure 4.14: Obstacle-avoidance based planner vs our planner for long-horizon minimum-time path planning.

The $15\,\mathrm{m} \times 20\,\mathrm{m}$ map for the long-horizon task demands a minimum time objective. The robot must navigate a series of blocks, requiring it to climb up and down multiple times and jump across gaps to reach the goal.

Our planner, equipped with the learned time cost predictor, selects waypoints that optimize the robot's traversal time by fully leveraging its locomotion skills. This includes climbing, descending, and jumping, enabling the robot to reach the goal in the shortest time possible. This contrasts with obstacle-avoidance-based planners, which fail to exploit the robot's capabilities and often choose less efficient paths.

Results. In the offline planning stage, the PRM planner samples nodes on the map to build the roadmap from start to goal. Due to the stochastic nature of node sampling, the roadmap may vary slightly between runs, leading to different planning results. To demonstrate consistency across multiple runs, we conducted five trials for each planner on every map with the same start and goal configuration. The quantitative results are summarized in Tables 4.15, 4.16, and 4.17.

planners	sampling time [s]	planning time [s]	node gen time [s]	GPU parallel time [s]	total cost [J]	sim success rate	sim energy cost (mean , std) [J]
run1_heu0_prediction_obs_avoid	7.365069	37.492034	0.413104	27.372249	6045.461426	5/5	1117.61068 , 17.1748486
run1_heu0_prediction	-	65.971144	0.388223	25.887549	2694.167969	5/5	617.49389 , 31.70661491
run1_heu2_prediction	-	4.688502	0.399186	26.566449	2953.543945	5/5	702.74462 , 14.24445537
run1_heu0_analytical	-	59.432103	0.416005	25.974054	**1181.308228**	5/5	**484.81123 , 13.98808419**
run1_heu2_analytical	-	**2.996511**	0.396118	26.343149	1226.66272	5/5	591.617928 , 25.74460571
run2_heu0_prediction_obs_avoid	6.809917	41.072785	0.373333	24.657029	7222.609375	4/5	1964.31642 , 225.8966273
run2_heu0_prediction	-	55.850572	0.365653	23.823958	2956.412598	5/5	589.845244 , 18.41726665
run2_heu2_prediction	-	11.145987	0.381073	23.753696	3074.038086	5/5	617.26564 , 97.38835059
run2_heu0_analytical	-	44.164505	0.3729	23.31425	**1191.670898**	5/5	**538.355502 , 14.6063455**
run2_heu2_analytical	-	**2.843557**	0.376543	24.14458	1377.993042	4/5	923.159698 , 91.79434988
run3_heu0_prediction_obs_avoid	7.437381	34.340959	0.359116	23.659318	6802.675293	5/5	1266.49494 , 37.58432477
run3_heu0_prediction	-	41.170231	0.336524	22.163544	2931.806152	5/5	537.647802 , 14.86487283
run3_heu2_prediction	-	3.682478	0.359798	22.294373	3090.266113	5/5	496.141058 , 10.25105254
run3_heu0_analytical	-	40.641494	0.358994	22.130782	**1168.391357**	5/5	**468.964508 , 4.80996963**
run3_heu2_analytical	-	**2.715519**	0.360799	22.267174	1178.665771	5/5	492.11942 , 10.39403093
run4_heu0_prediction_obs_avoid	7.211643	34.544921	0.426424	27.439591	6537.520508	5/5	1633.29464 , 413.2266703
run4_heu0_prediction	-	53.872778	0.40294	25.363995	2831.932373	5/5	575.26606, 15.50255111
run4_heu2_prediction	-	4.508556	0.400557	25.021257	2923.069092	5/5	578.90487, 26.29599849
run4_heu0_analytical	-	50.930645	0.375997	24.804066	**1208.767578**	5/5	**508.151144 , 3.71483913**
run4_heu2_analytical	-	**3.61741**	0.392811	25.522455	1307.242676	5/5	896.118652 , 198.9932168
run5_heu0_prediction_obs_avoid	7.062268	31.102097	0.34322	22.808169	6118.026855	5/5	1367.03046 , 26.99385528
run5_heu0_prediction	-	48.71955	0.341638	21.25141	2743.304199	5/5	544.924314 , 30.95721983
run5_heu2_prediction	-	4.639923	0.331372	21.554788	2923.774902	5/5	634.479478 , 33.60127517
run5_heu0_analytical	-	43.23159	0.322226	21.367988	**1171.791138**	5/5	**478.57961 , 5.810023723**
run5_heu2_analytical	-	**2.687212**	0.341858	21.824838	1209.395264	5/5	501.38478 , 11.10467187

Figure 4.15: Results of all the planners for minimum-energy path.

For every roadmap, we evaluated five different planners (Figures A.27 to A.29). These planners are as follows:

1. Obstacle-avoidance-based planner using predicted costs without heuristics.
2. Planner based on Dijkstra's algorithm using predicted costs without heuristics.
3. Planner based on weighted-A* using predicted costs.
4. Planner based on Dijkstra's algorithm using analytical costs without heuristics.

planners	sampling time [s]	planning time [s]	node gen time [s]	GPU parallel time [s]	total cost [s]	sim success rate	sim time cost (mean , std) [s]
run1_heu0_prediction_obs_avoid	6.764137	429.960779	1.112213	86.524157	35.593395	3/5	21.4119998 , 0.3454276712
run1_heu0_prediction	-	605.370998	1.097352	80.445199	**14.666033**	5/5	8.13599978 , 0.3444996528
run1_heu2_prediction	-	**4.149565**	1.101122	79.444236	21.093773	1/5	8.4519997 , 0.7995750521
run1_heu0_analytical	-	475.295385	1.165456	80.3006	22.006748	5/5	**6.8999998 , 0.6546750492**
run1_heu2_analytical	-	4.277445	1.167216	79.407941	24.199381	5/5	7.24799984 , 0.3559774869
run2_heu0_prediction_obs_avoid	10.126023	359.694463	1.091488	76.39692	34.94833	5/5	21.0719998 , 0.2142892811
run2_heu0_prediction	-	504.487967	1.110689	68.993006	**14.867062**	4/5	7.556 , 0.6735577184
run2_heu2_prediction	-	5.617786	1.038842	69.620253	17.451452	5/5	7.79599994 , 0.2558906063
run2_heu0_analytical	-	405.248195	1.046117	68.796832	20.460171	5/5	**6.55199992 , 0.1269646975**
run2_heu2_analytical	-	**4.204775**	1.081044	69.57852	21.115795	5/5	6.79599986 , 0.1818790054
run3_heu0_prediction_obs_avoid	8.700342	425.751988	1.116218	73.167889	36.973606	3/5	21.2719998 , 0.5080551505
run3_heu0_prediction	-	421.148908	1.037526	68.396129	**14.378777**	3/5	**6.63599988 , 0.2491584612**
run3_heu2_prediction	-	**3.604995**	1.086223	68.57247	15.371344	5/5	8.616, 0.5026728558
run3_heu0_analytical	-	350.973206	0.998893	68.51778	20.718349	4/5	6.65599982 , 0.2041568054
run3_heu2_analytical	-	3.609875	1.032441	67.990989	20.897057	5/5	6.73599988 , 0.3912543646
run4_heu0_prediction_obs_avoid	8.564244	330.963325	1.200262	87.430035	34.505718	5/5	20.8559996 , 0.579896427
run4_heu0_prediction	-	539.569606	1.115787	80.56769	**13.214941**	5/5	**6.75199996 , 0.2193627115**
run4_heu2_prediction	-	8.702871	1.160753	77.95767	15.614209	5/5	7.28799988 , 0.2862165691
run4_heu0_analytical	-	412.376716	1.1672	77.932932	21.036211	5/5	6.84399976 , 0.07797442786
run4_heu2_analytical	-	6.674961	1.102553	77.558369	21.990086	5/5	8.2 , 0.7557777451
run5_heu0_prediction_obs_avoid	6.072579	323.007521	1.041039	69.62835	36.487373	5/5	24.248 , 1.680987805
run5_heu0_prediction	-	415.953734	0.971693	64.904937	**14.948875**	5/5	7.53199988 , 0.5617117291
run5_heu2_prediction	-	5.378675	0.983319	66.555806	17.087477	5/5	7.9879996 , 0.4830319544
run5_heu0_analytical	-	330.664807	0.972282	64.74408	21.910887	4/5	**6.96399994 , 0.3415845746**
run5_heu2_analytical	-	4.092933	0.992965	65.895579	23.040199	5/5	8.00799974 , 0.8205608272

Figure 4.16: Results of all the planners for minimum-time path.

5. Planner based on weighted-A* using analytical costs.

Key Observations

- Across all scenarios, planners that leverage the robot's capabilities to plan paths over feasible obstacles consistently achieve significantly lower costs compared to the obstacle-avoidance-based planner.
- Weighted-A* planners, with a heuristic weight of 2.0, demonstrate significantly faster planning times. However, the use of this heuristic renders the planner inadmissible, resulting in suboptimal paths.
- For minimum energy tasks, planners using analytical costs outperform those using predicted costs. This is expected since the analytical cost functions are directly derived from controlled experiments, providing more accurate estimates

planners	sampling time [s]	planning time [s]	node gen time [s]	GPU parallel time [s]	total cost [s]	sim success rate	sim time cost (mean , std) [s]
run1_heu0_prediction_obs_avoid	13.334338	247.512418	0.929701	69.395621	34.631222	5/5	20.332 , 0.3545701623
run1_heu0_prediction	-	520.813787	0.95633	63.455633	**24.833687**	2/5	**16.6899995 , 0.2899141338**
run1_heu2_prediction	-	**10.29323**	0.881329	64.312246	40.846451	5/5	21.5879996 , 0.4975138671
run1_heu0_analytical	-	223.792691	0.876677	61.961835	67.991745	5/5	20.6639998 , 1.413534687
run1_heu2_analytical	-	10.878828	0.94841	64.197083	75.066086	5/5	23.0159996 , 1.659180191
run2_heu0_prediction_obs_avoid	17.66699	183.023088	0.922353	62.587272	41.095451	3/5	24.6719998 , 0.712404461
run2_heu0_prediction	-	369.946766	0.827661	58.393747	**27.67602**	4/5	**17.1879996 , 0.8801821505**
run2_heu2_prediction	-	10.16941	0.884645	61.074148	39.474228	4/5	22.6039998 , 0.4746368738
run2_heu0_analytical	-	176.693988	0.816605	59.933736	70.30175	3/5	21.768 , 0.476571086
run2_heu2_analytical	-	**10.026379**	0.833632	59.866402	72.542587	4/5	22.544 , 0.3465256123
run3_heu0_prediction_obs_avoid	14.834605	227.034913	0.891837	65.195186	37.500057	4/5	21.0519998 , 0.1162755606
run3_heu0_prediction	-	393.616172	0.893552	61.613133	**24.822813**	5/5	**15.2439996 , 0.4265205833**
run3_heu2_prediction	-	10.574775	0.887121	59.726874	42.819279	2/5	25.31499975 , 1.087183197
run3_heu0_analytical	-	199.859574	0.830832	58.858118	69.542839	4/5	22.716 , 1.470945274
run3_heu2_analytical	-	**10.092356**	0.856402	60.670114	70.452202	4/5	23.3759992 , 1.806454285
run4_heu0_prediction_obs_avoid	17.883963	263.552201	0.86645	61.019721	35.740757	5/5	20.7199998 , 0.1788856898
run4_heu0_prediction	-	389.522811	0.862448	57.492367	**25.121288**	5/5	**14.7119996 , 1.043321993**
run4_heu2_prediction	-	9.933816	0.874329	58.721039	39.798618	4/5	21.3119998 , 0.5264220132
run4_heu0_analytical	-	241.539825	0.864316	57.63257	69.695915	5/5	20.2119996 , 0.4748895998
run4_heu2_analytical	-	**9.89526**	0.864016	58.352638	71.981522	5/5	22.0919996 , 0.8386413965
run5_heu0_prediction_obs_avoid	13.285764	209.995985	0.899886	65.197599	33.835461	5/5	20.132 , 0.1480540442
run5_heu0_prediction	-	391.637623	0.84352	60.071077	**23.312933**	3/5	**15.3759998 , 2.949080768**
run5_heu2_prediction	-	**10.131573**	0.920936	61.322621	36.341213	5/5	21.8599996 , 0.6634760131
run5_heu0_analytical	-	194.907423	0.86433	60.058396	68.05191	5/5	19.9879998 , 0.4448816966
run5_heu2_analytical	-	10.654175	0.904479	63.593254	69.239555	4/5	20.2199998 , 0.3088688557

Figure 4.17: Results of all the planners for long-horizon minimum-time path.

for such scenarios. Both approaches, however, yield paths with much lower costs than the obstacle-avoidance-based planner.

- For minimum time tasks, planners using predicted and analytical costs perform comparably, as both methods produce similar time estimates. The resulting paths effectively exploit the robot's capabilities to minimize traversal time.
- For long-horizon minimum time tasks, planners using predicted time costs consistently outperform those using analytical costs. This highlights the value of learned cost predictors in navigating complex environments with varying obstacle types and long planning horizons.

Ground Truth Validation

To validate the planned paths, we executed them using the robot in the Isaac Gym simulator and compared the resulting ground truth costs with the estimated costs. While the ground truth costs often differed from the estimated costs (due to factors like stochastic policies and unmodeled dynamics), the trends in total costs across planners were consistent. Importantly, these differences did not affect the feasibility of the planned paths, demonstrating the robustness of the S3D-OWNS framework.

Conclusion

These results underscore the effectiveness of blending learning-based policies with a high-level planner that utilizes learned cost predictors to optimize the desired objectives. The S3D-OWNS framework consistently generates optimal paths, even for long-horizon tasks, enabling efficient navigation and robust decision-making in complex environments.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

We conclude that the S3D-OWNS framework, by leveraging the robot's capabilities to plan optimal paths, enables efficient navigation in unstructured and cluttered environments. Depending on the task requirements, the robot can select paths that either minimize energy consumption—extending operational time—or prioritize faster traversal to complete tasks efficiently. This adaptability makes the framework particularly suitable for industrial applications, such as factory settings, where efficiency and cost-effectiveness are critical.

A significant strength of the framework lies in its modularity. Each module can be independently replaced or upgraded with more sophisticated counterparts, making it versatile and future-proof. For example, the planner, locomotion policy, or cost predictors can be substituted with improved versions without altering the overall framework, allowing for continuous performance enhancement.

The simplicity of the planner is another key advantage. By operating in a reduced state space, the planner avoids the computational burden of reasoning in high-dimensional spaces. This simplicity is particularly beneficial in multi-agent scenarios where multiple robots must coordinate efficiently across a shared environment. The ability to integrate powerful policies ensures high-level planning is both robust and efficient.

Despite its strengths, the framework has certain limitations. Its reliance on the

cost predictor's accuracy is a notable challenge. For instance, due to the unreliable performance of the trained success predictor, it was not utilized in our experiments. Training neural networks to estimate edge costs requires careful consideration of all possible scenarios that could arise during planning. Simulating these scenarios during data collection ensures a diverse dataset, reducing the likelihood of encountering out-of-distribution data during deployment.

The framework's reliance on the simulation environment for data collection is another limitation. The accuracy of the collected data depends on the fidelity of the simulator. In our case, NVIDIA's Isaac Gym provided a highly efficient platform with parallel GPU acceleration, significantly speeding up data collection. Without this, the process would have taken weeks or months.

Looking ahead, these challenges can be addressed by improving the diversity of datasets, exploring better cost predictor architectures, and integrating higher-fidelity simulations to enhance real-world applicability.

5.2 Future Works

The following directions are proposed to extend and improve upon the S3D-OWNS framework:

- **Path Smoothing:** Introduce post-processing steps to optimize the paths generated by the sampling-based planner. This refinement could ensure smoother paths, reducing unnecessary turns and improving traversal efficiency.
- **Advanced Neural Architectures:** Employ more powerful neural networks, such as transformers, for cost predictors. These architectures could enhance the accuracy of success estimators, making them more reliable and robust, thereby adding an extra layer of safety to the planning module.
- **Larger Local Patches:** Experiment with larger local patches to reduce the need for excessive sampling near obstacle edges. This approach could increase the number of feasible paths without requiring additional sampling at gaps or block edges.
- **Diffusion Policies for Path Generation:** Explore diffusion-based policies to generate multiple paths from start to goal. These policies could create diverse

path candidates, which can then be evaluated using cost predictors to identify the optimal path.

- **Integrated Cost Learning:** Extend the framework to train cost predictors or value functions directly alongside the locomotion policy. Simultaneous training could reduce reliance on separate datasets and ensure better alignment between the learned costs and the policy.
- **Sim-to-Real Transfer:** Enhance sim-to-real transfer by using techniques such as DAgger and distillation processes. While previous works, such as [1], demonstrated successful sim-to-real transfer for generalist policies, long-horizon tasks require more accurate state estimators to prevent drift over extended navigation sequences.
- **Real-World State Estimation:** Improve state estimation by integrating external sensors, such as motion capture (MoCap), to refine state feedback during planning. Initially, the pipeline could leverage external sensing before transitioning to fully onboard state estimation.
- **Scene Understanding and High-Level Commands:** Incorporate scene-understanding methods and Large Language Models (LLMs) to enable natural language interfaces. This could allow users to define start and goal locations using voice or text commands, broadening the framework's applicability in complex environments.
- **Multi-Agent Coordination:** Extend the framework to multi-agent planning scenarios, where multiple robots navigate and collaborate within the same environment. This could involve integrating decentralized planning techniques or communication protocols to optimize coordination.
- **Exploration of Diverse Applications:** Apply the framework to diverse fields such as search and rescue, agriculture, and warehouse automation. Each of these domains offers unique challenges that could further validate and improve the framework's robustness.

The S3D-OWNS framework lays a solid foundation for efficient and adaptable robotic navigation. By addressing the aforementioned directions, the framework can be further refined and expanded, enabling robots to tackle increasingly complex tasks

in real-world scenarios.

Appendix A

Additional Results

A.1 Trends in Dataset

A.1.1 2D Energy Plots

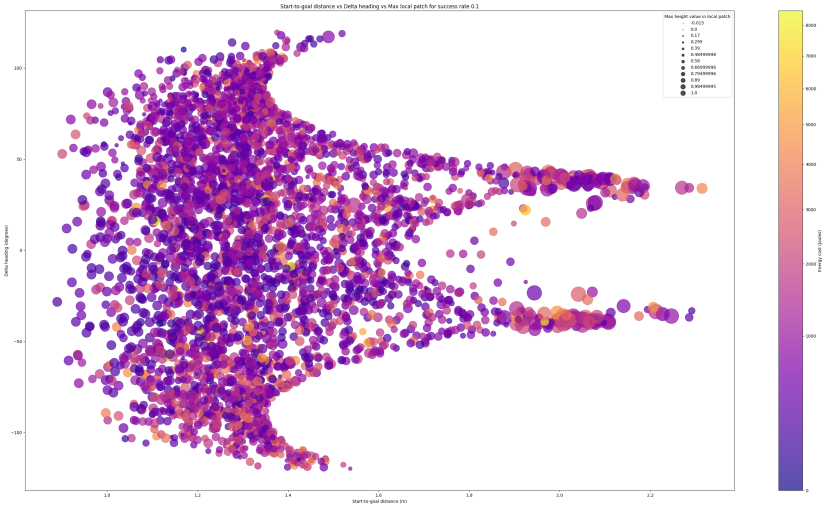


Figure A.1: **Success Rate 0.1: Energy distribution as per success rate for the hurdle world cross dataset**.

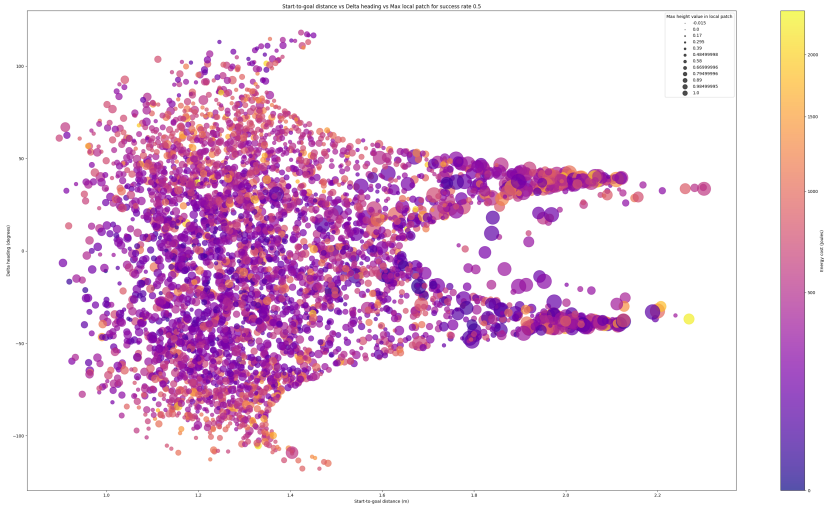


Figure A.2: **Success Rate 0.5: Energy distribution as per success rate for the hurdle world cross dataset.**

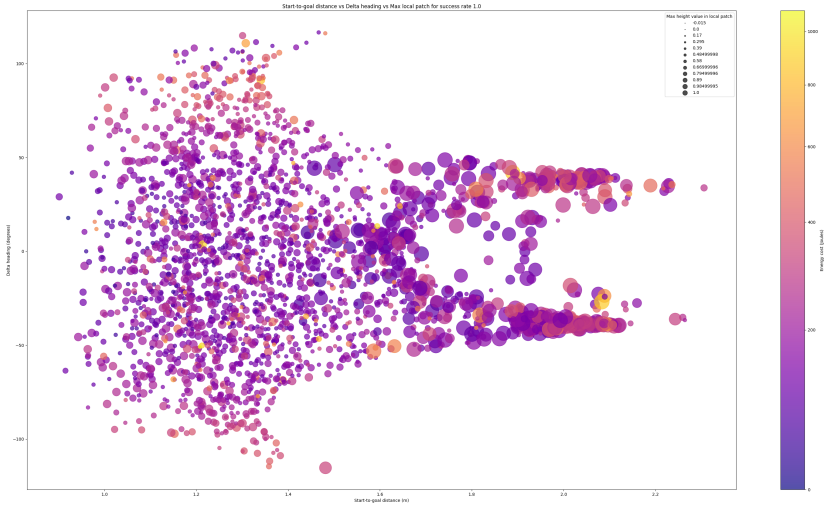


Figure A.3: **Success Rate 1.0: Energy distribution as per success rate for the hurdle world cross dataset.**

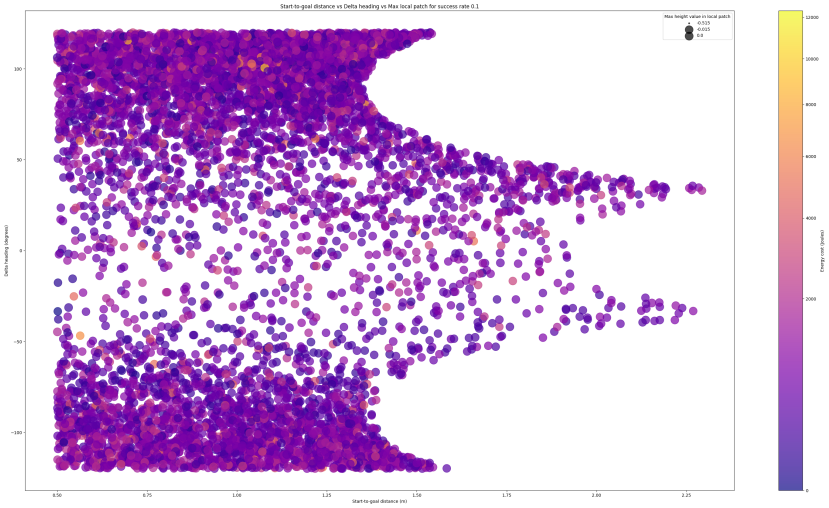


Figure A.4: **Success Rate 0.1: Energy distribution as per success rate for the gap world cross dataset**.

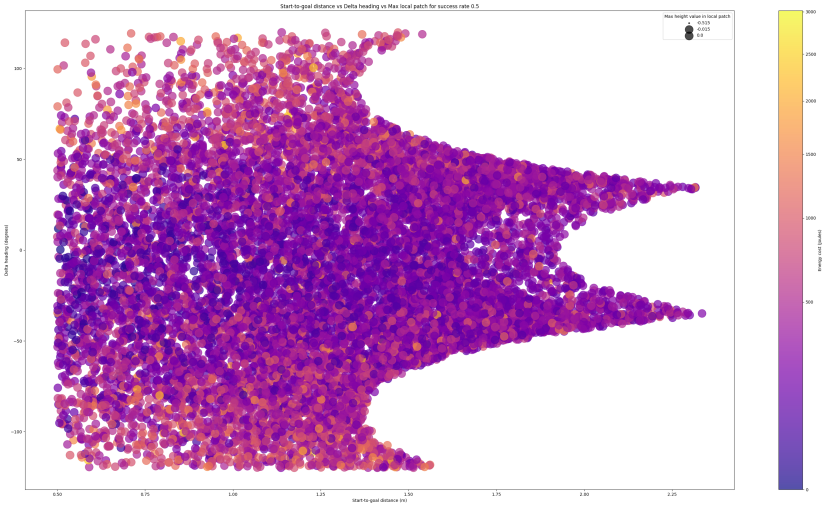


Figure A.5: **Success Rate 0.5: Energy distribution as per success rate for the gap world cross dataset**.

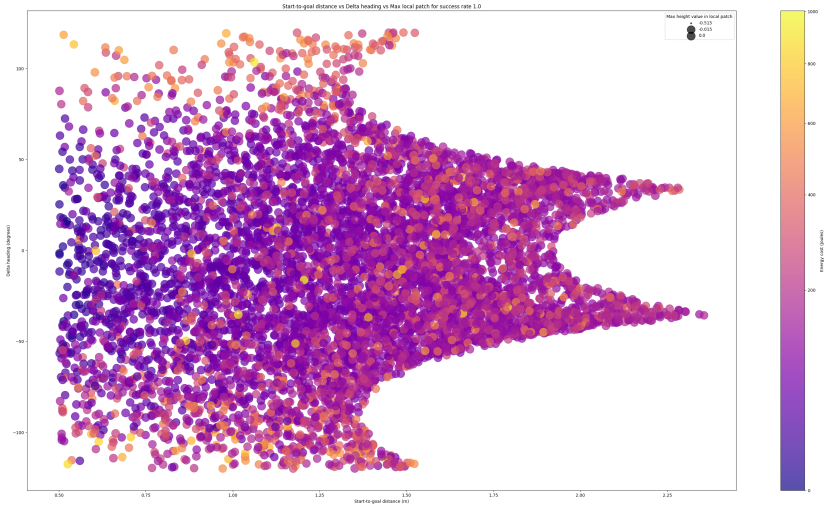


Figure A.6: **Success Rate 1.0: Energy distribution as per success rate for the gap world jumping dataset**.

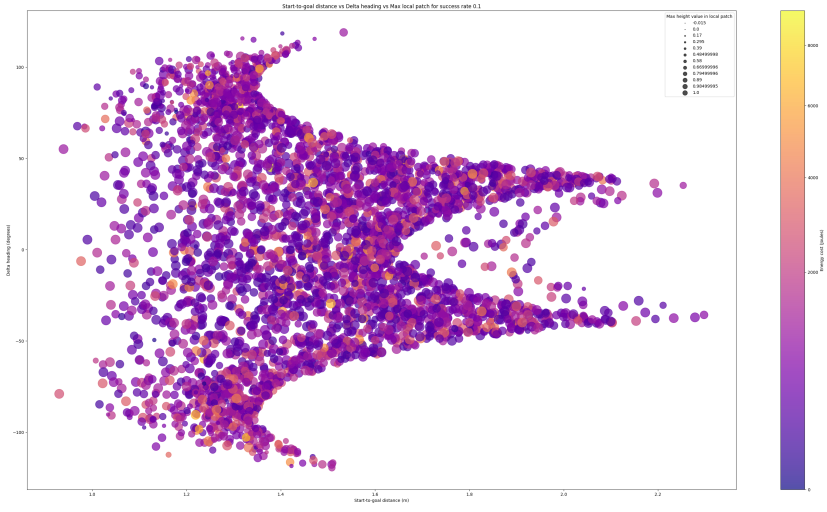


Figure A.7: **Success Rate 0.1: Energy distribution as per success rate for the block world going up dataset**.

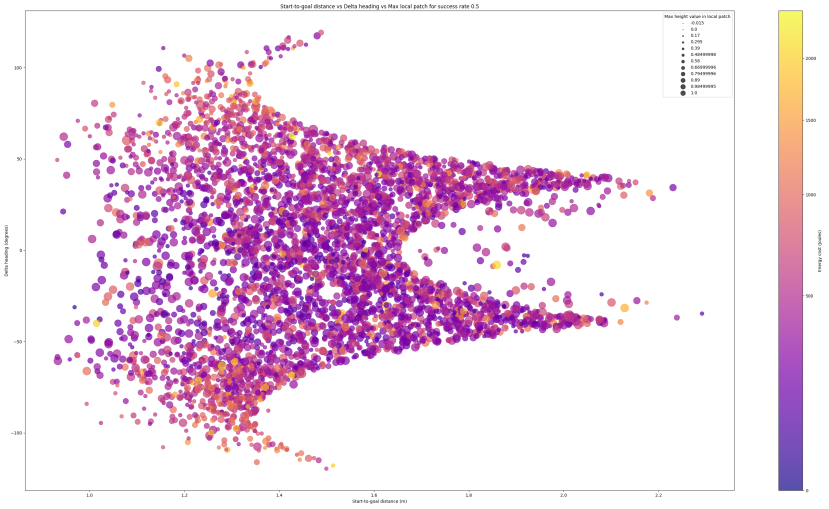


Figure A.8: **Success Rate 0.5: Energy distribution as per success rate for the block world going up dataset.**

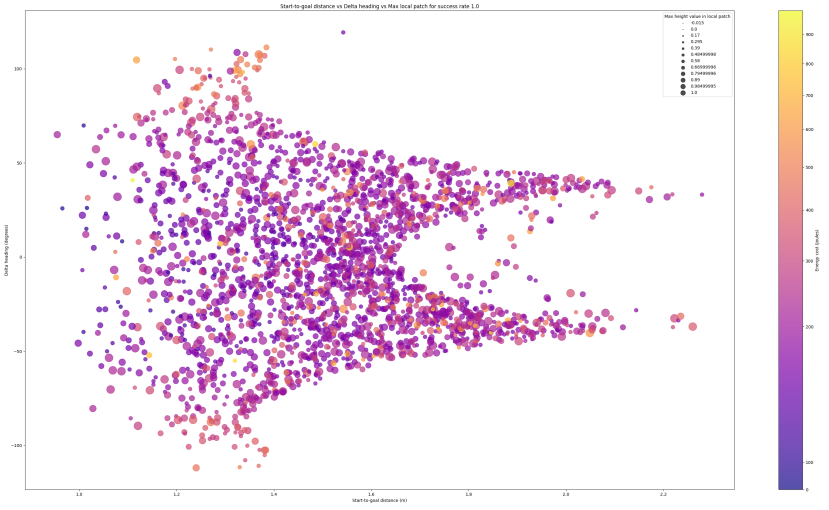


Figure A.9: **Success Rate 1.0: Energy distribution as per success rate for the block world going up dataset.**

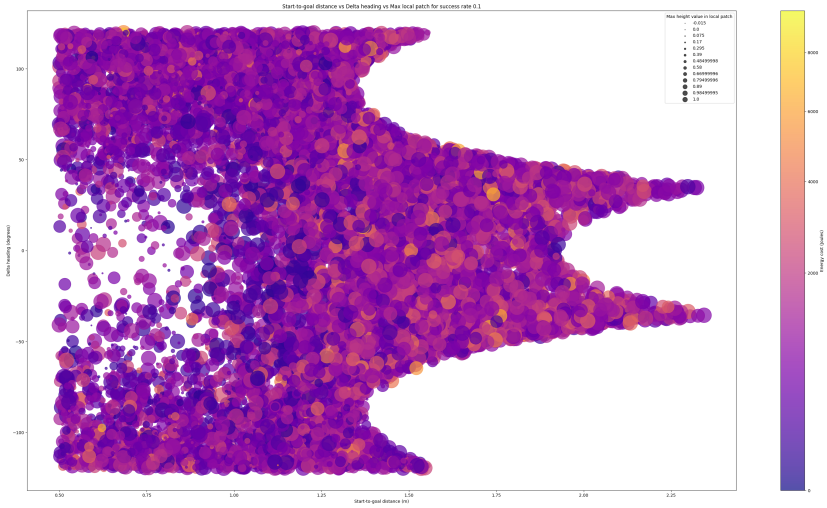


Figure A.10: **Success Rate 0.1: Energy distribution as per success rate for the block world going down dataset**.

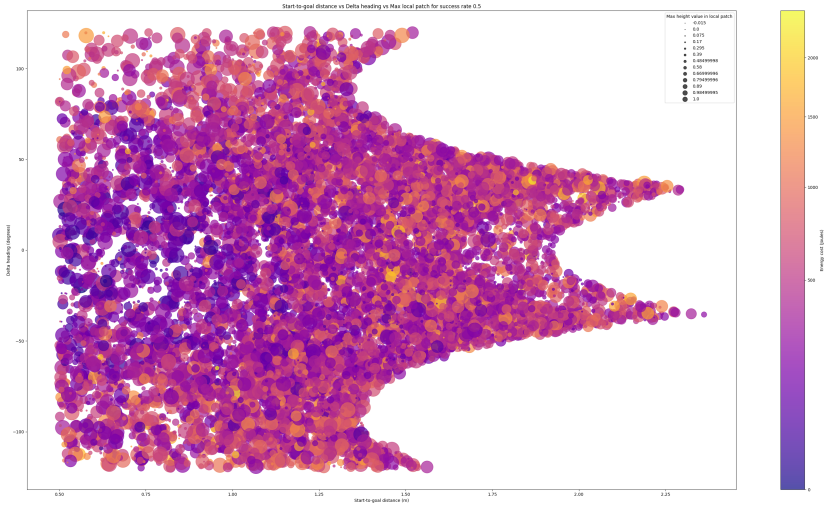


Figure A.11: **Success Rate 0.5: Energy distribution as per success rate for the block world going down dataset**.

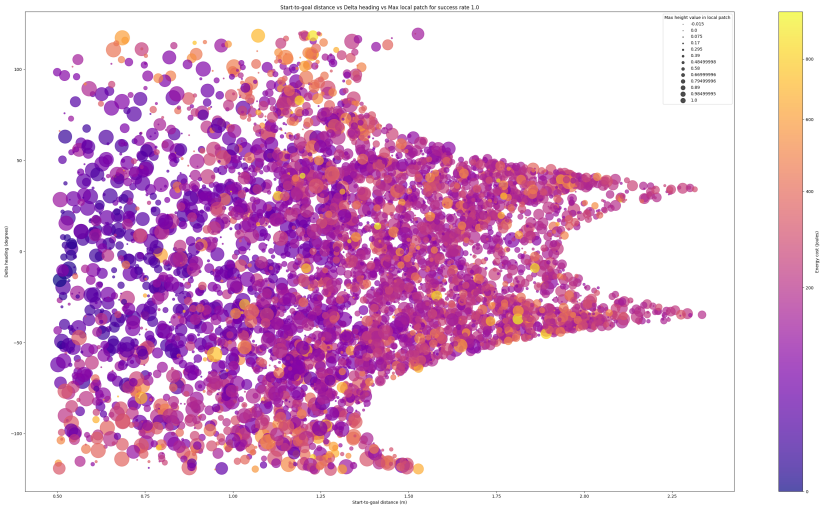


Figure A.12: **Success Rate 1.0: Energy distribution as per success rate for the block world going down dataset**.

A.1.2 3D Energy Plots

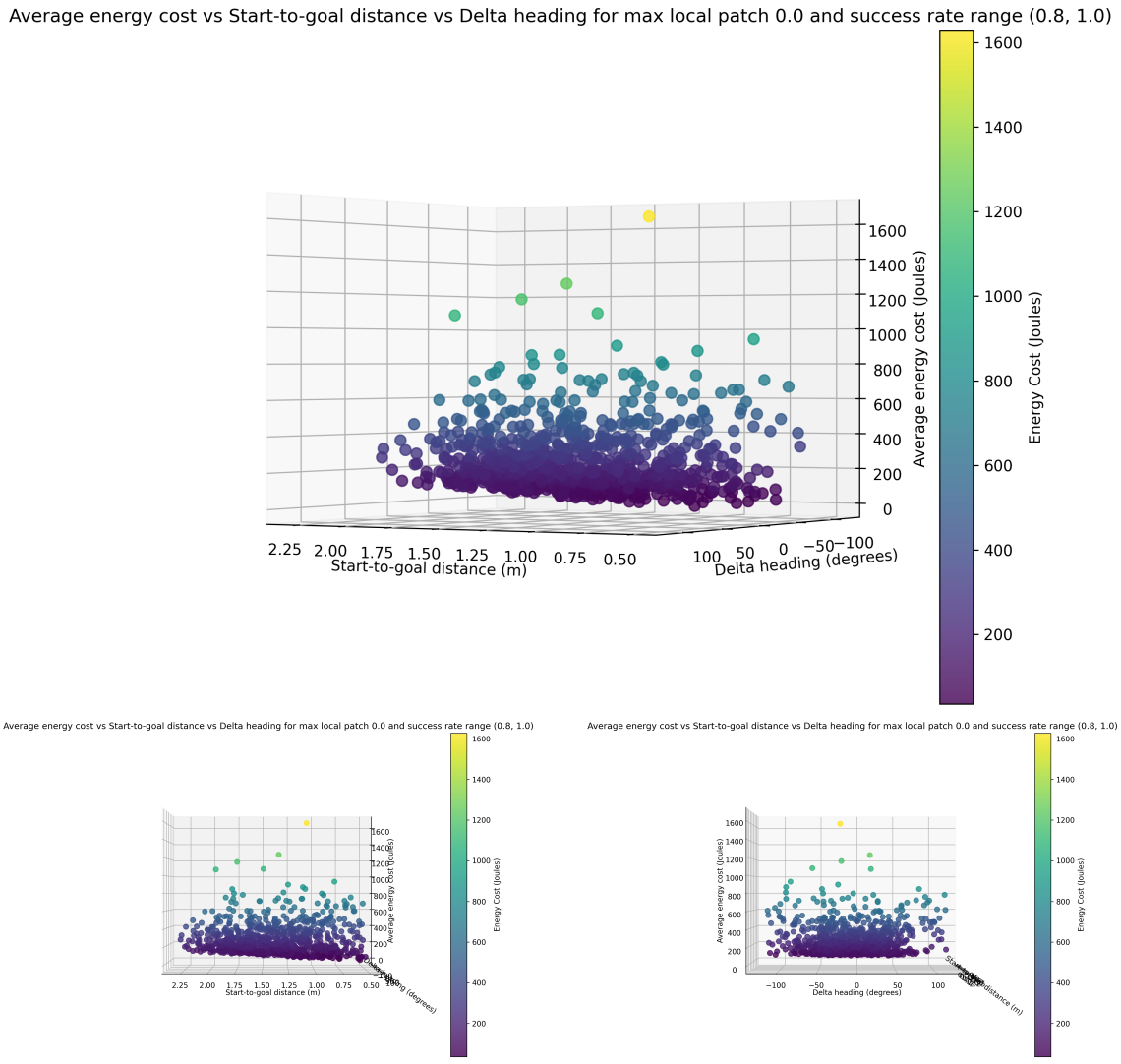


Figure A.13: **Average energy cost vs Start-to-goal distance vs Delta heading for max local patch 0.0 and success rate range (0.8, 1.0). Different views of the same data are presented to highlight the relationships between variables.**

A.1.3 2D Time Plots

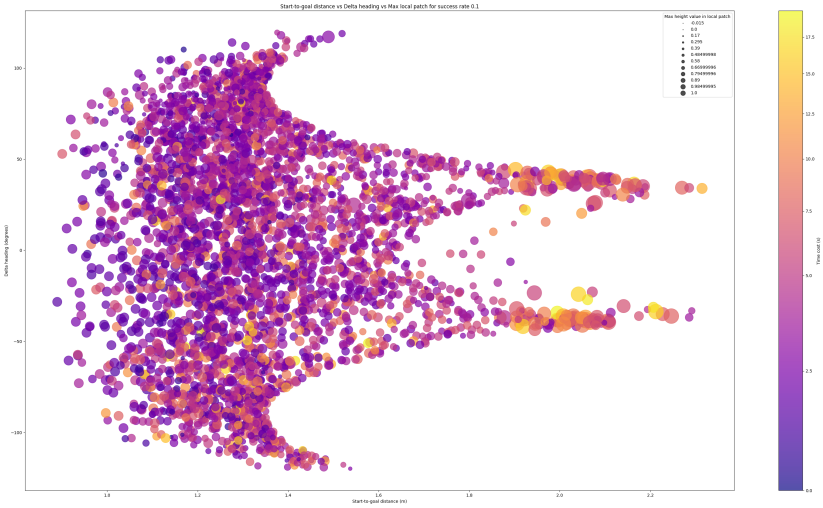


Figure A.14: **Success Rate 0.1: Time distribution as per success rate for the hurdle world cross dataset**.

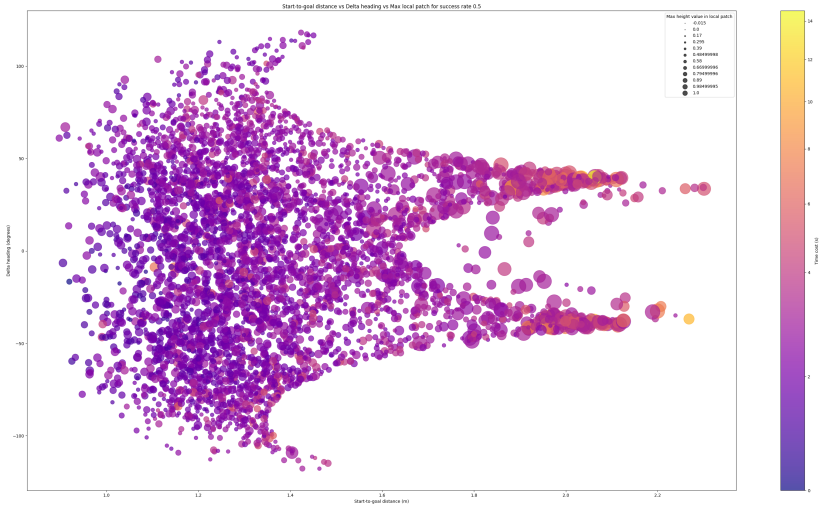


Figure A.15: **Success Rate 0.5: Time distribution as per success rate for the hurdle world cross dataset.**

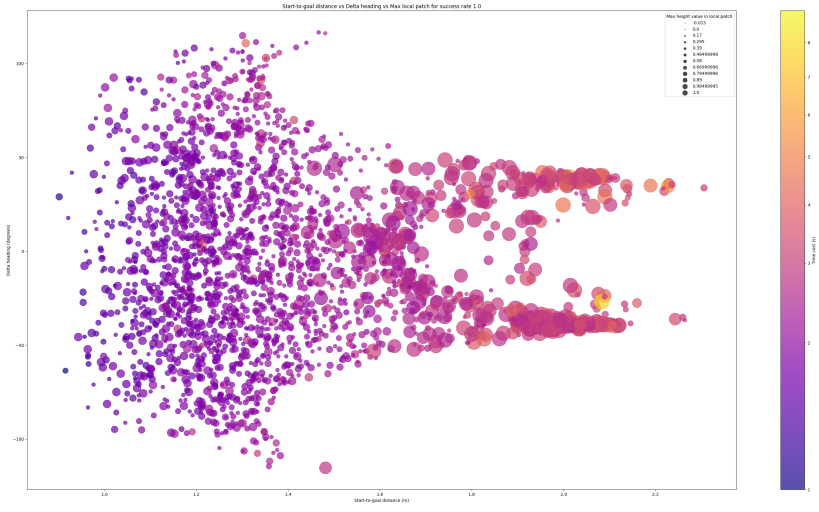


Figure A.16: **Success Rate 1.0: Time distribution as per success rate for the hurdle world cross dataset.**

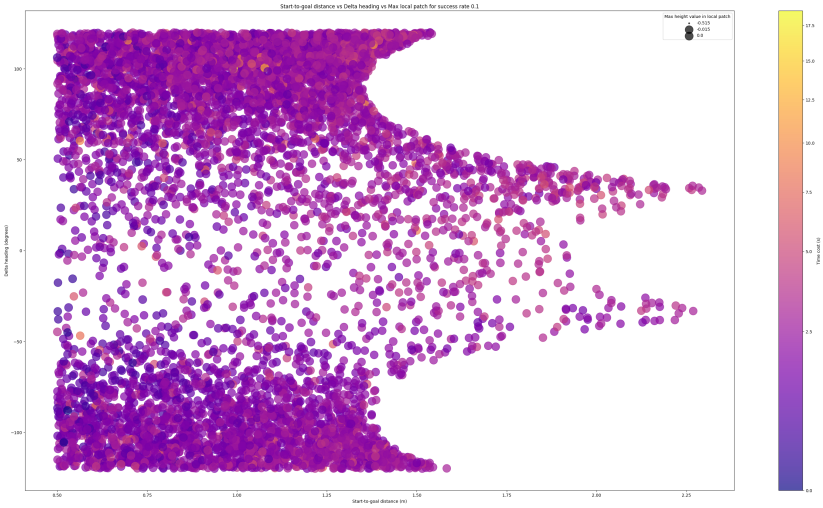


Figure A.17: **Success Rate 0.1: Time distribution as per success rate for the gap world cross dataset.**

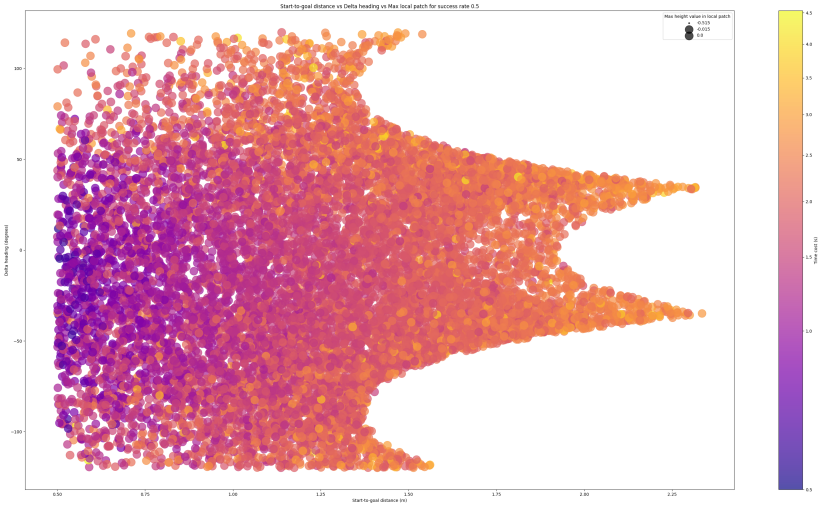


Figure A.18: **Success Rate 0.5: Time distribution as per success rate for the gap world cross dataset.**

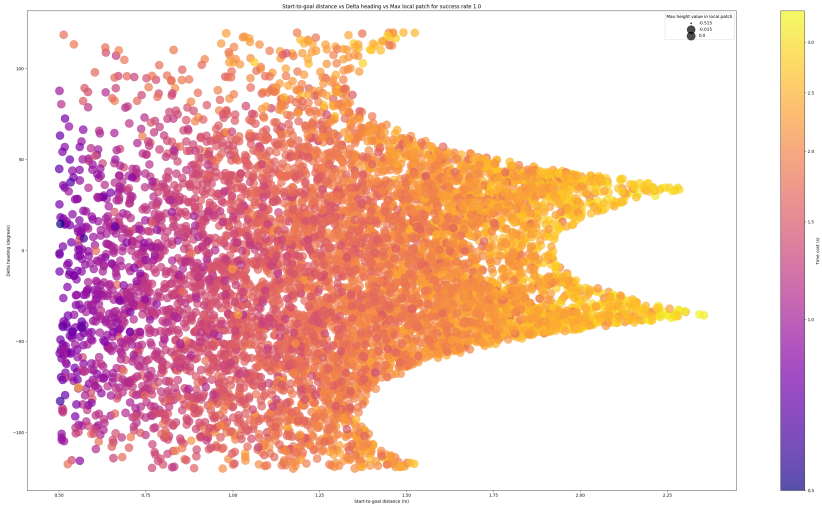


Figure A.19: **Success Rate 1.0: Time distribution as per success rate for the gap world jumping dataset**.

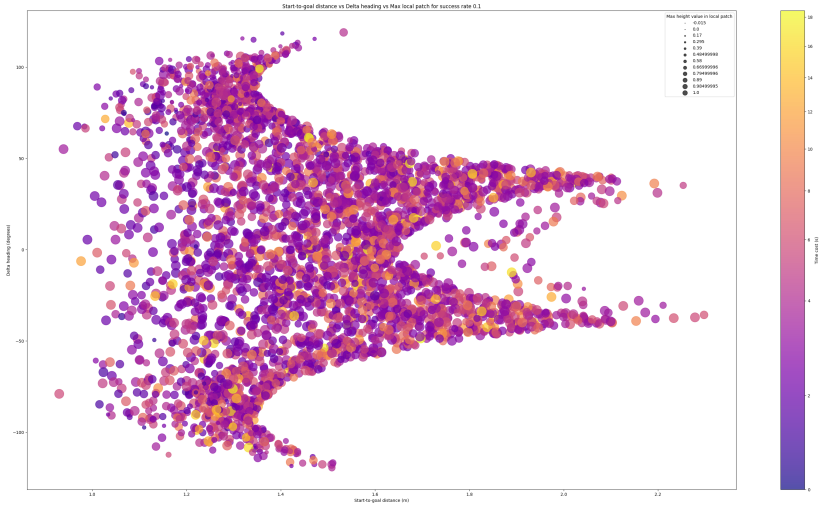


Figure A.20: **Success Rate 0.1: Time distribution as per success rate for the block world going up dataset**.

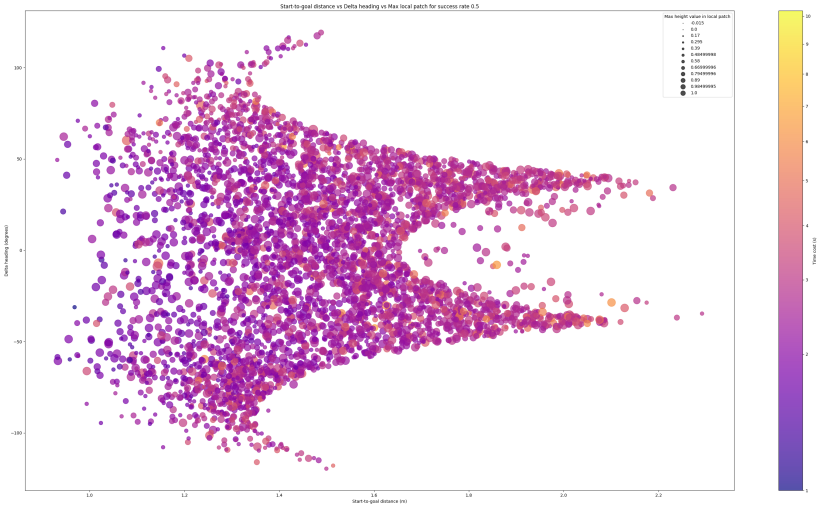


Figure A.21: **Success Rate 0.5: Time distribution as per success rate for the block world going up dataset.**

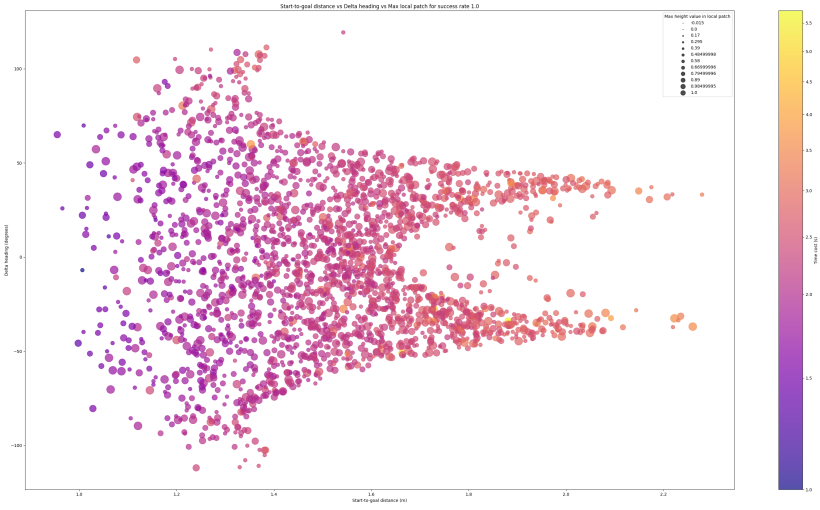


Figure A.22: **Success Rate 1.0: Time distribution as per success rate for the block world going up dataset.**

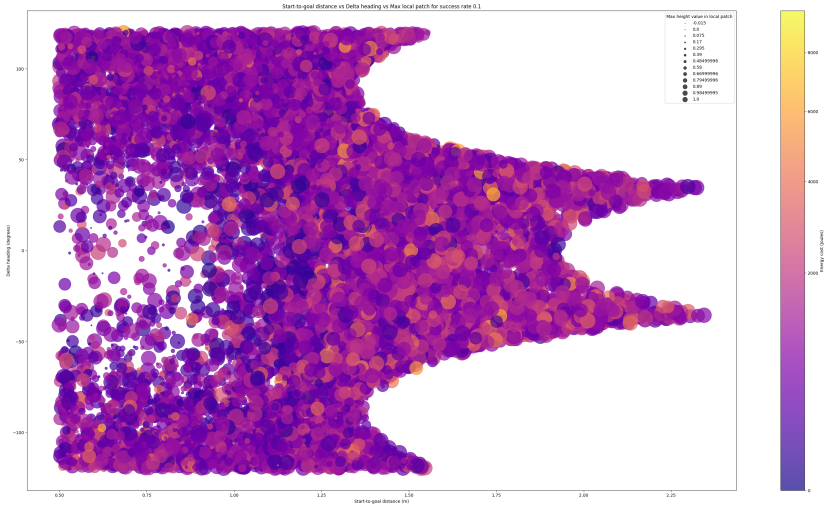


Figure A.23: **Success Rate 0.1: Time distribution as per success rate for the block world going down dataset.**

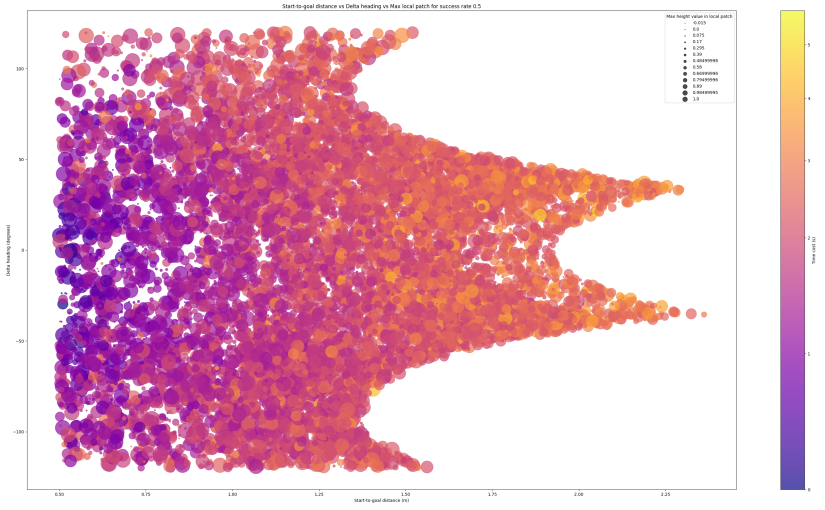


Figure A.24: **Success Rate 0.5: Time distribution as per success rate for the block world going down dataset.**

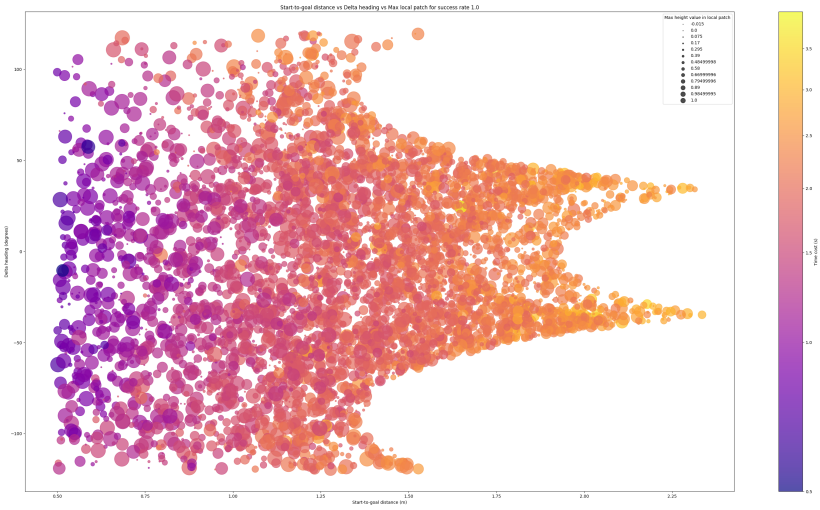


Figure A.25: **Success Rate 1.0: Time distribution as per success rate for the block world going down dataset**.

A.1.4 3D Time Plots

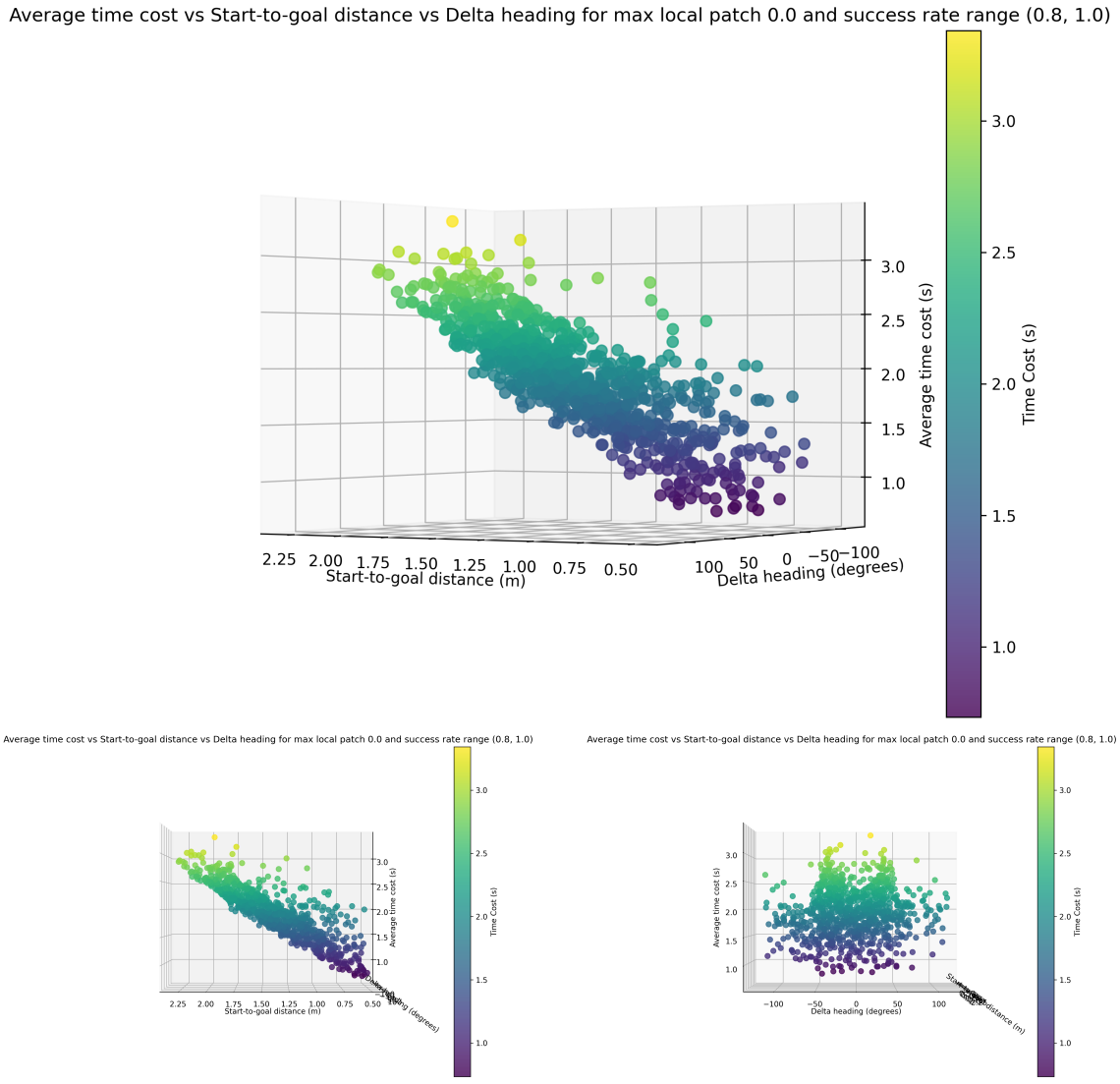


Figure A.26: **Average time cost vs Start-to-goal distance vs Delta heading for max local patch 0.0 and success rate range (0.8, 1.0). Different views of the same data are presented to highlight the relationships between variables.**

A.2 Different Planners

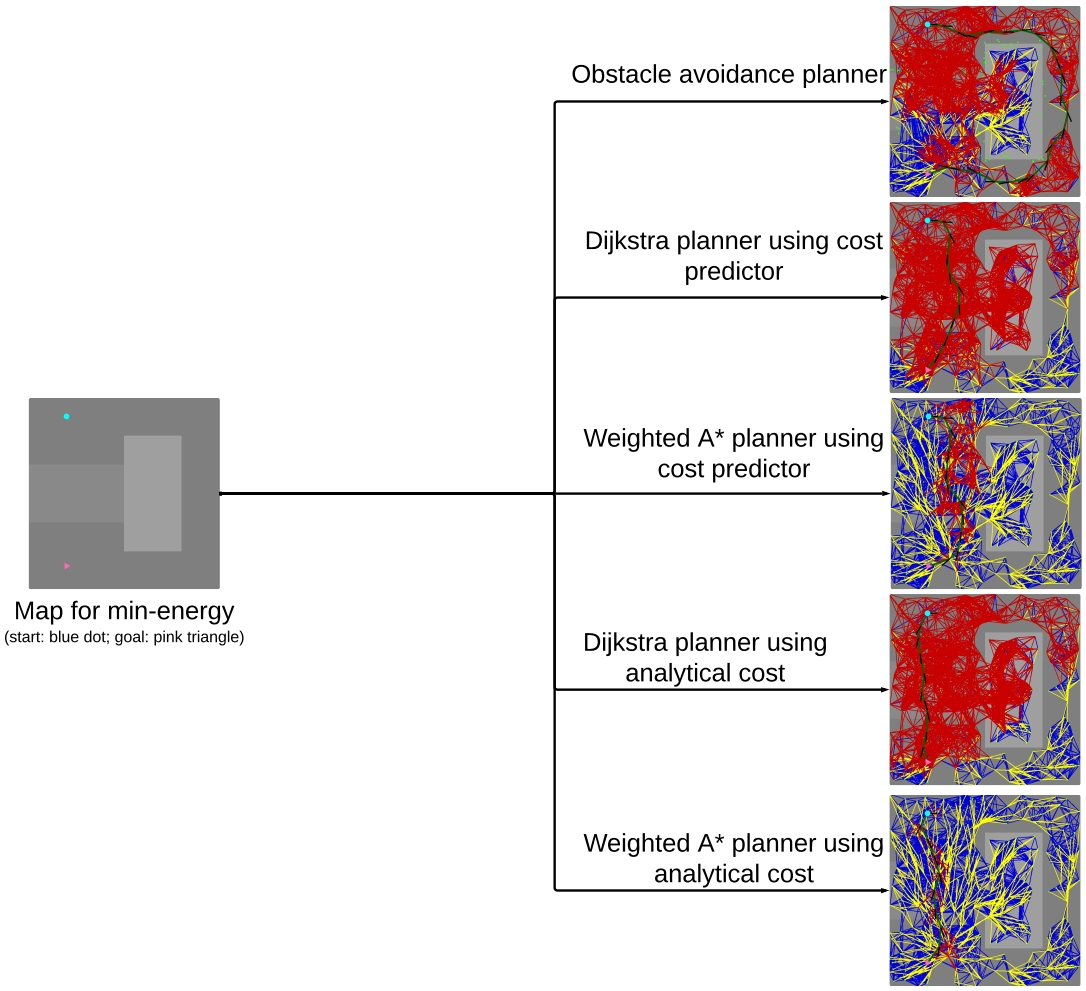


Figure A.27: **Different planners for minimum-energy path**.

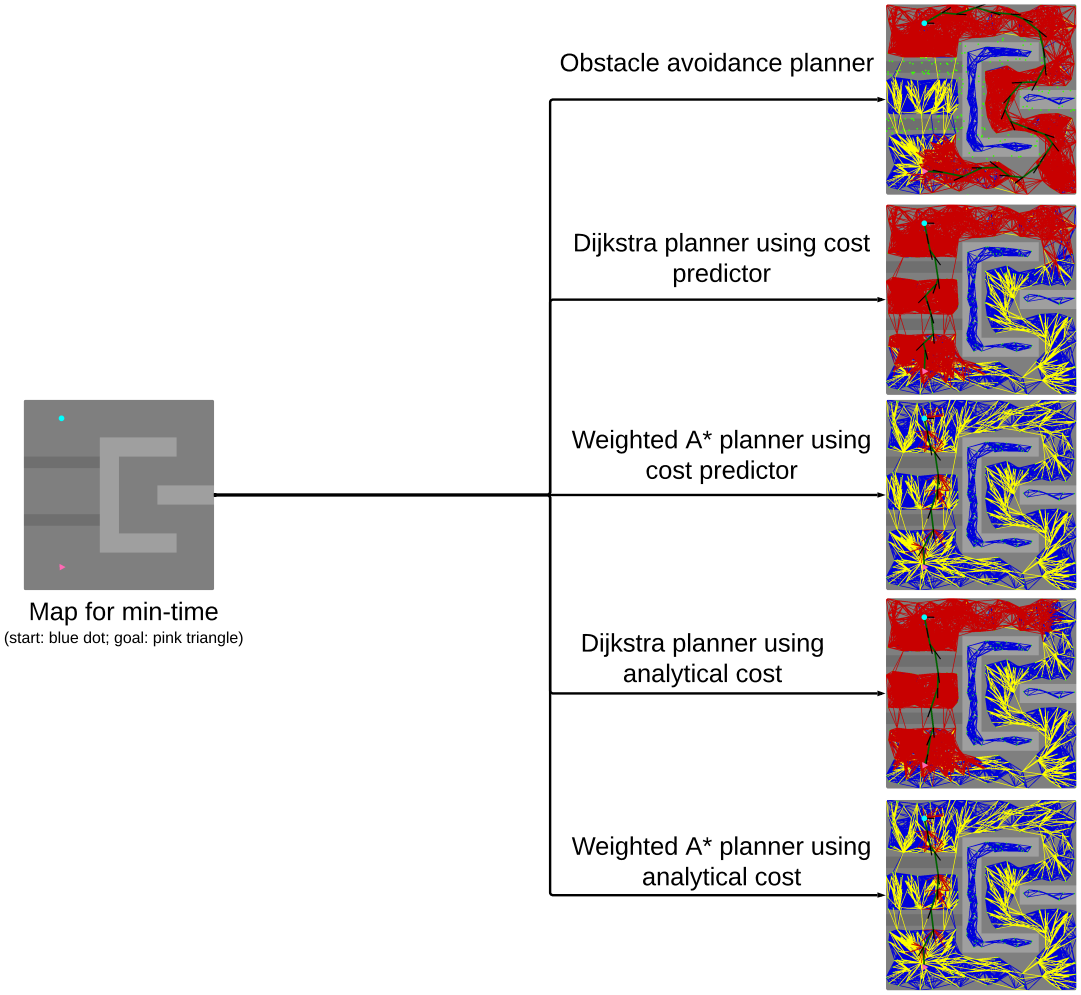


Figure A.28: **Different planners for minimum-time path**.

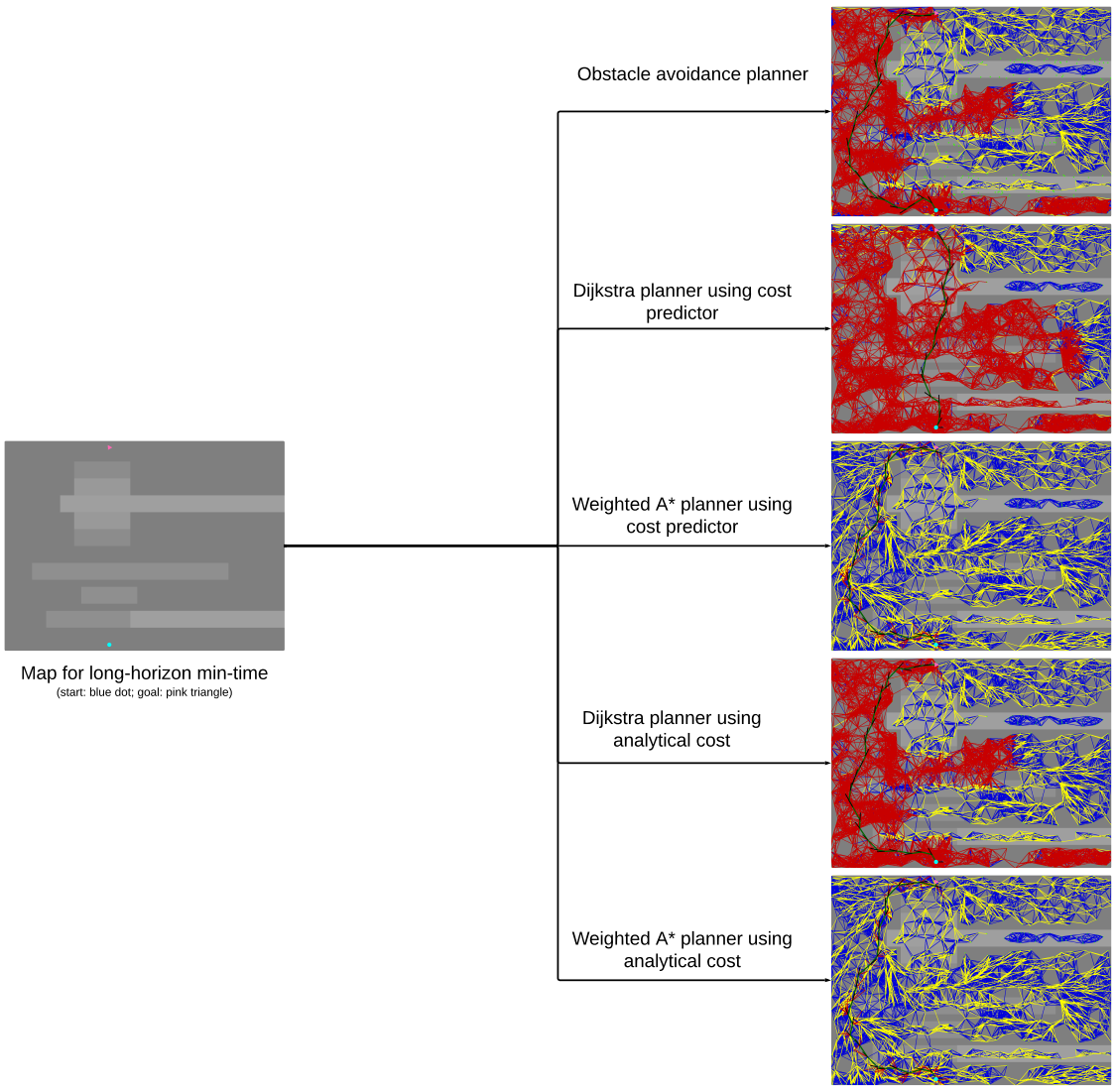


Figure A.29: **Different planners for long-horizon minimum-time path**.

Bibliography

[1] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023. 1.0.1, 1.0.3, 2.0.1, 3.2.1, 3.2.1, 5.2

[2] Jérôme Guzzi, R. Omar Chavez-Garcia, Mirko Nava, Luca Maria Gambardella, and Alessandro Giusti. Path planning with local motion estimations. *IEEE Robotics and Automation Letters*, 5(2):2586–2593, 2020. doi: 10.1109/LRA.2020.2972849. 3.2.2

[3] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots, 2023. URL https://arxiv.org/abs/2306.14874. 1.0.1, 2.0.1, 3.1

[4] Amanda A Howard, Sarah H Murphy, Shady E Ahmed, and Panos Stinis. Stacked networks improve physics-informed training: applications to neural networks and deep operator networks, 2023. URL https://arxiv.org/abs/2311.06483. 3.2.1

[5] Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011. ISBN 9780374275631 0374275637. URL https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7. 3.1

[6] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. URL https://arxiv.org/abs/2108.10470. 1.0.1

[7] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning, 2022. URL https://arxiv.org/abs/2205.07802. 3.2.1

[8] Pascal Roth, Julian Nubert, Fan Yang, Mayank Mittal, and Marco Hutter. Viplanner: Visual semantic imperative learning for local navigation, 2024. URL https://arxiv.org/abs/2310.00982. 2.0.3

[9] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2022. URL https://arxiv.org/abs/2109.11978. 2.0.1

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3.2.1

[11] Maria Stamatopoulou, Jianwei Liu, and Dimitrios Kanoulas. Dippest: Diffusion-based path planner for synthesizing trajectories applied on quadruped robots, 2024. URL https://arxiv.org/abs/2405.19232. 2.0.3

[12] Lorenz Wellhausen and Marco Hutter. Rough terrain navigation for legged robots using reachability planning and template learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6914–6921, 2021. doi: 10.1109/IROS51168.2021.9636358. 2.0.3

[13] Bowen Yang, Lorenz Wellhausen, Takahiro Miki, Ming Liu, and Marco Hutter. Real-time optimal navigation planning using learned motion costs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9283–9289, 2021. doi: 10.1109/ICRA48506.2021.9561861. 2.0.2, 2.0.3

[14] Fan Yang, Chen Wang, Cesar Cadena, and Marco Hutter. iplanner: Imperative path planning, 2023. URL https://arxiv.org/abs/2302.11434. 2.0.3

[15] Wentao Zhang, Shaohang Xu, Peiyuan Cai, and Lijun Zhu. Agile and safe trajectory planning for quadruped navigation with motion anisotropy awareness. *arXiv preprint arXiv:2403.10101*, 2024. 1.0.1

[16] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109:43–76, 2019. URL https://api.semanticscholar.org/CorpusID:207847753. 3.2.1