Intra-Robot Replanning and Learning for Multi-Robot Teams in Complex Dynamic Domains

Philip Cooksey

CMU-RI-TR-19-21

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

April 2019

Thesis Committee:
Manuela Veloso, Chair
Maxim Likhachev
Stephen Smith
Daniel Borrajo Millán, Universidad Carlos III de Madrid

Keywords: intra-robot replanning, rationale-driven plan, centralized planner, rationale, constraints, multi-robot teams, complex dynamic domains

To my family and friends.

Abstract

In complex dynamic multi-robot domains, there is a set of individual robots that must coordinate together through a centralized planner that inevitably makes assumptions based on a model of the environment and the actions of the individual. Eventually, the individuals may encounter failures, because the centralized planner's models of the states and actions are incomplete and the assumptions it makes are incorrect. In this thesis, we address the problem of what an individual robot must do when faced with such failures and can no longer execute the plan generated by the centralized planner.

While previous work has exclusively explored centralized approaches or decentralized approaches for dynamic multi-robot problems, it lacks the combination of a centralized approach with intelligent planning individuals, whom are often found in decentralized approaches. In centralized approaches, the focus has been on removing the need for replanning through conditional planning and policy generations, on hierarchical decomposition to simplify the multi-robot problem, or on predicting the informational needs of teammates. In decentralized approaches, the focus has been on improving auctioning algorithms, task decomposition, task assignment, and policy generation. In this thesis, I contribute a novel intra-robot replanning algorithm for the individual robots that autonomously handle failures with a set of pre-defined plans. To make local replanning feasible, I introduce a rationale-driven plan that provides the reasoning behind the choices made by the centralized planner. The intra-robot replanning algorithm then has a choice of how to fix the plan, given the set pre-defined plans and provided rationales, or of invoking the centralized planner. We can improve this process by learning scores for the pre-defined plans that are used by the intra-robot replanning algorithm to improve the performance of the robot.

This thesis is motivated by my previous work with individual robot replanning and centralized planners and the inability of their individual robots to handle failures. With autonomous underwater vehicles (AUVs), their approach to failures is to rise to the surface of the ocean, message the centralized controller, and wait for a new plan. With the Small-Size League soccer robots, their approach to failures is to continue, blindly executing a failing team plan, because taking at least some course of action, even if currently failing, can be better than inaction in an adversarial domain. Of course, continuing to execute a failing plan is an inadvisable approach, but the individual soccer robots lacked the necessary individual intelligence to fix the problem. These domains share the common thread of relying on the centralized planner to provide them with a new solution to handle failures. And, in doing so, they act inefficiently, are slow to react, and oftentimes have a higher cost in regard to the team's performance.

In this thesis, we describe our rationale-driven plan which details the reasoning for the actions and parameters chosen within the plan. We then explain our intra-robot replanning algorithm which uses the rationale-driven plan to replan locally. We evaluate the work with multiple domains in different environments to provide evidence of the effectiveness and generality of our approach. We then describe our method for learning and improving the intra-robot replanning algorithm. Lastly, we discuss possible future work that can expand upon the work in this thesis.

Acknowledgments

My journey to completing my thesis has been supported by my family, my advisors, and my friends, all whom have helped me achieve a goal I once thought impossible. There are not enough words to detail the gratitude I have towards them in my heart, and there is no way I can do them justice here, but all the same, I will try.

Foremost, I want to thank my parents Gary and Sherry for believing in me and supporting me even when they did not fully understand my aspirations. I want to thank my sister Jamie for always inspiring me to be better and achieve more, and my brother Daniel for being there when I needed him the most. My family members are my biggest supporters and they helped me more than I will ever know.

Of course, there would be no thesis without my advisor, Manuela Veloso, who has supported me and challenged me throughout my PhD. She gave me the freedom to explore my own path, and I thank her for the opportunity to work in CORAL and conduct research on the Small-Size soccer robots. I would also like to thank my other thesis committee members: Maxim Likhachev, Stephen Smith, and Daniel Borrajo Millán for their feedback on my proposal, this thesis document, and my defense.

I want to thank Kanna Rajan for all the support he has provided me and the support he continues to give. Kanna took a chance on me during my undergraduate and helped introduce me to the world of robotics and artificial intelligence. He also helped expand my thesis by connecting me to João Sousa and the Laboratório de Sistemas e Tecnologia Subaquática, where they let me work with their autonomous underwater vehicles; I cannot thank them enough. Thank you Kanna for believing in me.

I want to thank Bill Head at the Undergraduate Research Opportunity Center in my undergraduate university for meeting with me and agreeing to support my passions and research interests. And, I want to thank him for taking on the even more challenging task of teaching me how to be a better researcher, writer, and person. Thank you Bill for supporting my research and constantly challenging me to be better.

I want to thank Kate Lockwood, my undergraduate advisor, for seeing potential in me and supporting my research. Kate Lockwood introduced me to Kanna, without whom my whole journey would have been different.

I want to thank Josh Gross who, on the very first day of his class, asked which students wanted to get a PhD. He asked me to meet with him and kick-started my journey by pushing me to meet with Kate and Bill, which eventually lead me to meet with Kanna, and then Manuela. Thank you Josh for taking the time to care about your students and taking the time to care about my future.

Of course, I want to thank all my past and present labmates. They have been there when it has been tough and when it has been amazing. I would especially like to thank my CMDragon teammates. Competing in the RoboCup world competitions was stressful but winning 1st place and two 2nd places in the Small-Size League was highly rewarding.

Lastly, I want to thank my girlfriend Lydia Jahl. She has been the rock that has balanced my ever-changing mental scales. She has been my chief editor and one of my biggest supporters. Thank you Lydia for always supporting me and for being patient, loving, and kind.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> “There are plenty of teams in every sport that have great players and never win titles. Most of the time, those players aren't willing to sacrifice for the greater good of the team. The funny thing is, in the end, their unwillingness to sacrifice only makes individual goals more difficult to achieve. One thing I believe to the fullest is that if you think and achieve as a team, the individual accolades will take care of themselves. *Talent wins games, but teamwork and intelligence wins championships.* ”
>
> *~Michael Jordan, former professional basketball player [Jordan, 1994, pg. 20-24]*

Each member of a team must willingly sacrifice for the welfare of the team, using *teamwork*, but they also need to intelligently contribute to the team, using *intelligence*. Balancing teamwork with motivated intelligent players is the key to winning championships, and championships are a microcosm for human intelligent behavior: coordination, communication, dynamic movement, adversarial, reacting, planning, replanning, and learning. Of course, one thesis can only tackle a small portion of such a strong artificial intelligence (AI) problem, i.e., a general human intelligence problem.

In this thesis, we focus on the individual robots and their ability to replan locally, quickly, and proactively towards achieving team goals, while being instructed by some centralized team planner. The centralized team planner wants the team to be successful in achieving the team goals, and to do so, it generates a plan for each individual robot that it considers to be achievable. However, the models used to generate such a plan are often incomplete and inaccurate. Given imperfect models, the centralized planner creates a plan that oftentimes does not hold valid throughout the entire execution of the plan. Often, a failure occurs at the individual level, leaving the individual robot with a frequently overlooked choice of *how* to handle such a failure.

In the literature, the standard approach to *how* is to have the robot wait for a new plan or perform a standard set of actions to keep the robot safe, e.g., an autonomous underwater vehicle (AUV) surfaces and signals for help, a Mars rover stops and waits while calling Earth, or a soccer robot continues executing until a new plan is received. This approach has the implicit assumption that the environment will not change for the worse while the robot is waiting for a new plan. However, if we remove that assumption and consider environments that continue to change, making the situation worse for the robot, we must consider the consequences of *how* the individual robot handles failures. Given the possibility of such an adversarial environment and the time delay

between the robot and the centralized planner, an intelligent robot has a choice in the face of a failure: blindly follow the provided plan, stop and wait for a new plan, or as this thesis introduces, replan locally to handle the failure and then continue with the previously provided plan. In our work, we consider replanning as the robot selecting from a set of pre-defined plans (policies) that can be used to handle some failure case, where it is often a many-to-one mapping of pre-defined plans to failure. We are investigating the choices of the individual robot beyond the provided instructions of the provided plan while remaining true to the centralized planner's mission and goals.

Our approach is to take the benefits of centralized team planning – tightly coordinated, global goal-oriented, and predetermined joint actions – and combine them with the benefits of intelligent individuals – locally reactive, adaptable, and computationally simpler. The centralized team planning approaches gather global information and produce plans that can be tightly coordinated and globally optimal, if feasible. However, the individual robots often lack the intelligence and/or independence to handle local dynamic changes without waiting for the centralized approach. Conversely, decentralized approaches rely on the intelligence of the individuals to reduce computation and to find locally feasible (if not locally optimal) solutions. However, the decentralized approaches often lack the agility to produce tightly coordinated plans thereby lacking many benefits of centralized team planning. In many ways, decentralized methods are full of *talented* players but those players struggle to develop the necessary *sacrificing* teamwork. Our approach is to combine a centralized planner with intelligent robots that can choose how and when to proactively replan locally, if the situation requires the robot to do so.

In our experience, replanning is inevitable. Current methods of modeling complex dynamic domains are imperfect and simplified due to the complexity of such domains. The centralized planner's ability to generate a reliable multi-robot plan is hampered by imperfect models and the problem is only further compounded by the multi-robot dimensionality. The consequences appear when the individual robot executes its plan and there is a mismatch between the modeled state and the current executed state, invalidating conditions in the plan provided by the centralized planner. Furthermore, in removing the assumption of a safe environment, we should assume that the environment may get worse for the individual robot if it does nothing proactive. Therefore, replanning locally is inevitable if we want to have reliable performance in our robot teams using current methods of modeling complex dynamic domains.

The challenge for the individual robot is to remain compliant with the provided plan, or more generally with the reasoning of the centralized planner. The centralized planner makes many choices during the process of generating the team plan and in general has some reason for selecting the actions and their parameters within the plan. Therefore, our approach adds the centralized planner's reasoning into the plan, and ultimately provides that reasoning to the individual robots executing the plan. The replanning algorithm we introduce uses the centralized planner's reasoning while replanning locally to ensure that the individual robot is not counterproductive to the objectives of the centralized planner.

1.1 Thesis Question

This thesis seeks to answer the question,

> How can an *individual* robot, within a centralized controlled team, *locally* handle failures and changes in dynamic environments while remaining *compliant* to the *reasoning* of the *centralized controller*?

The **individual** robot is the fundamental unit of any team and ultimately, the individual robot contends with failures and unforeseen changes in the environments caused in part by the complexity of modeling the dynamics of the robot and environment. We argue that many such failures can be solved **locally** by an individual robot, i.e., without the intervention of the centralized controller, and doing so improves the overall performance of the team.

The challenge for the individual robot is to remain **compliant** with the provided plan, but more generally with the **reasoning** of the **centralized controller**. The centralized controller has its own models and reasons for choosing the action within the plan. And, in order for the individual robot to improve the performance of the team, the actions chosen by the individual robot must not be counterproductive to the objectives of the centralized controller.

1.2 Approach

Our approach in this thesis involves individual robots proactively replanning, i.e., selecting an alternative plan from a set of pre-defined plans, given they are already provided with an informative plan, a rationale-driven plan. The approach consists of attaching the centralized planner's rationale, i.e., the constraints and reasoning used by the centralized planner in selecting the actions for each individual robot, to the team plan to generate the rationale-driven plan. This enables the individual robots to replan locally when there are changes in the environment that result in a plan failure, while remaining compliant with the centralized planner's objectives. The individual robots monitor the execution of their rationale-driven plan and proactively choose how and when to replan locally and when to ask for a new plan from the centralized planner. Failures come in many different forms and as such there are often many different methods that can be used to handle each failure, a many-to-one mapping. During the execution of the rationale-driven plan, the individual robot chooses which pre-defined plan to use to re-enable the failing rationale or condition(s) of the plan. Learning helps inform the individual robots to make better choices by learning the most effective pre-defined plan given the current state of the environment, thereby providing the individual robot with the knowledge of when to use a particular pre-defined plan.

1.2.1 Rationale-Driven Plan

In order to provide the individual robots with information to enable local replanning, we introduce the centralized planner's rationale in the plan representation. The rationales represent the reasoning behind the choices of the centralized planner. Rationales are preconditions, effects, goals, subgoals, constraints, or any choice made by the centralized planner that reduces the search space during the planning process and ultimately produces the final plan. In other words, the rationales we introduce are the model's constraints on the actions of the robot and on the environment, are

the decisions (and assumptions) made by the centralized planner, and are any external constraints placed on the planning process, e.g., for efficiency, human-user preference, or safety. In general, the rationale that the centralized planner uses during the process of creating the final plan remains valid during the execution of the final plan.

1.2.2 Intra-Robot Replanning

Ultimately, we are investigating the individual robot's role in replanning locally while being instructed by some centralized planner. However, the complex dynamic domains that our robots operate in make replanning, similar to planning, a challenging pursuit. There are many ways to handle a failure for the single robot within a team:

- Arguably the least effective method is the blind executor, where the robot continues executing its plan as if nothing has changed in the environment. This method can lead the robot to continuously attempt an infeasible action. As a counterpoint, in adversarial domains like robot soccer, blindly executing the plan might be a good option if the centralized planner cannot generate an alternative plan, e.g., continuing an attempt to kick the ball into the goal, that is now blocked, can be better than stopping and holding the ball until the centralized planner generates a new plan.

- The robot waits for the centralized planner to generate a new plan. This is often a very safe approach in many domains, but this assumes the environment does not continue to worsen, e.g, in robot soccer, the opposing robot gets closer to stealing the ball as the robot waits. This assumption is unrealistic and waiting can often be detrimental, therefore we do not make this assumption. Furthermore, this can take an extensive amount of communication time in certain domains, e.g., environments with satellite communication like ocean frontier tracking or space exploration.

- Our approach, where the robot chooses to replan locally to fix the local failures and attempts to reachieve the failed rationale. This provides a more proactive approach to handling problems for the individual in adversarial environments.

We assume the robot has multiple methods (replan policies) that can enable rationales should they change and make the current plan infeasible. An example rationale is that a particular path is chosen for a robot to take because it is *clear* and becomes false if the path is not *clear* during execution. A few examples of replan policies would be to clean the path to make it clear again, find another path to avoid the unclear path, if such an action is applicable, or the fall-back replan policy of informing the centralized planner that the path is no longer clear and the current plan is invalid. The question we tackle in this thesis is *how* does the robot choose what replan policy to execute given a state of the world and a rationale-driven plan. The provided rationales enable the robot to choose what replan policies should be used to re-enable a failed rationale. The individual robot then executes one of these replan policies and attempts to fix the failure and continue with the plan provided by the centralized planner. In picking one policy from the applicable ones, we sort them based on some pre-defined ordering or score. Learning a score, rather than using a pre-defined ordering, can be more beneficial when trying to optimize the selection of one replan policy from the applicable ones.

1.2.3 Learning

The issue of *how* to choose what replan policy to use in complex dynamic domains also includes *when* to pick a certain replan policy to handle the failure. There are oftentimes multiple policies that can handle a particular failure or re-enable a rationale because there are often multiple ways of handling a situation. So assuming there are multiple choices, *when* does the robot select a particular replan policy? Learning provides the individual robot with an opportunity to gather experience about the different replan policies and over time, learn to pick the best one that optimizes some metric or score.

Our method takes some information about the environment state and produces a score for each replanning policy. The score depends on the metric that we are trying to optimize for in the environment and the score aligns with the objective we are attempting to improve through replanning. For example, we may want to learn the probability (the metric) that a policy will succeed to improve the success rate (the objective) of the individual or we may want to learn the true cost (the metric) of the replan policy to reduce the total cost (the objective) of replanning. For our method, we use neural networks as our representation of the state to score function. The inputs are a subset of the state variables of the individual robot's environment. The output is the number representing a score for a given replanning policy for that particular state. We train the neural networks using historical data of when the robot executed a particular replan policy. Therefore, each individual robot has one neural network for each replan policy that produces a number, or score, based on the current environment.

1.3 Contributions

The key contributions of this thesis are as follows:

- Formalize the rationale-driven plan that includes information on how the actions of the plan are selected, constrained, or processed by the centralized planner, i.e, the reasoning of why the centralized planner chooses the particular actions and their associated parameters within the rationale-driven plan.
- Introduce algorithms for generating the rationale-driven plan.
- Introduce a novel intra-robot replanning algorithm that:
  - Determines the rationales of the plan that are no longer valid,
  - Determines the replan policies that are capable of repairing the rationales that are no longer valid,
  - Sorts the applicable replan policies according to some predetermined ordering or by a learned score for each replan policy,
  - Executes one by one the sorted replan policies until the rationale becomes valid or a new plan is requested from the centralized planner.
- Demonstrates the benefit of learning value functions for each replan policy based on the state of the environment in order to improve the sorting of the replan policies to further improve the performance of replanning locally.

- Formalize domains that use centralized planning and intra-robot planning to handle the dynamic, perhaps adversarial, nature of the environment.
- Evaluate intra-robot replanning in different dynamic domains showing improvement over a traditional purely centralized planner approach. These domains include robot soccer, autonomous underwater vehicles, and a formal planning description domain.

1.4 Thesis Outline

Section 2.6 provides a quick overview of the chapters in this thesis. For a more detailed outline, Chapter 2 provides a running example that breaks down the core elements of this thesis. The example provides a comprehensive understanding of the different problems being addressed in this thesis, and how the different chapters combine together in order to tackle the problem of intra-robot replanning within a centralized controlled multi-robot team.

Chapter 2

A Running Example: A Guide to the Thesis

This chapter runs through an example domain and highlights the key issues within this thesis. The example breaks down the problem into the different elements that were briefly discussed in the introduction chapter. This thesis involves different aspects of centralized planning, replanning, and individual adaptation that are needed to tackle the overall challenge of intra-robot replanning, and the running example illustrated in this chapter provides a primary example of the problem, how it was handled, and how the simpler solutions can be generalized to the methods provided in this thesis. Lastly, we summarize how each part of this example relates to specific parts of this thesis and how the key ideas implemented into this example problem are generalized to handle similar problems.

2.1 Domain Description and Chapter Outline

In this section, we describe the robot soccer domain used in our example. The RoboCup Small-Size League (SSL) is a multi-robot domain consisting of teams of six robots that play soccer in a highly dynamic and adversarial environment. Overhead cameras track the positions of the ball and each robot on the field, which are fed into a centralized computer shared by both teams, see Figure 2.1. The shared vision system referred to as SSL vision also detects and provides the field markers such as the half line, goal line, and goal box to each team [Zickler et al., 2009]. Each team must autonomously coordinate their robots in real time over radio waves, with the ultimate goal of manipulating the ball and scoring more goals than the opposing team to win. Both team coordination and individual robot skills are important aspects of this domain. In this running example, we focus on centralized planning and individual robot execution using opponent-aware ball-manipulation individual skills.

To plan tractably in a domain as complex and time-sensitive as robot soccer, one can separate the team planning aspect of the problem, e.g., to whom and where the robot controlling the ball should pass, from the execution of the plan, e.g., how to pass/move the ball to the chosen teammate [Browning et al., 2005]. This example focuses on the division of work between team planning and individual robot execution. We specifically address the problem of a robot that is tasked with moving the ball to a specific target location under opponent pressures, i.e., opponents blocking or attempting to steal the ball from the assigned robot. This task and location is assumed to be provided by a centralized team planner, but the robot has the opportunity to evaluate different

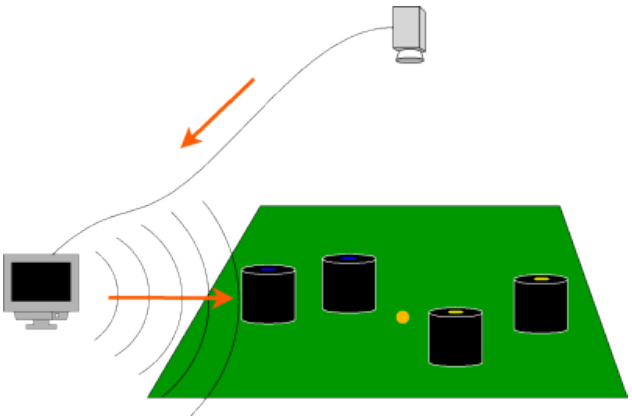

Figure 2.1: The flow of data from cameras to a centralized computer that controls the robots by radio waves [Tucker, 2010].

Figure 2.2: One of CMDragon's robots, with a ball touching its dribbler bar (horizontal black cylinder). The dribbling bar can be spun to put backspin on the ball for semi-control. Designed and built by Michael Licitra.

methods of achieving its task of moving the ball, given its ball-manipulation skills.

To manipulate the ball, most teams in SSL have converged to similar mechanisms: a kicker to impart momentum on the ball, and a dribbler bar to dribble the ball, see Figure 2.2. Their ball-manipulation skills thus depend on these mechanisms, and the optimal skill depends on the state of the opponents. Kicking the ball directly to its target is a highly accurate method of moving the ball, provided no opponents are nearby to steal the ball before the kick or to intercept it before it reaches its destination. Alternatively, the robot can dribble the ball to a better location before kicking, which is less reliable in the absence of opponents, due to the risk of losing the ball while dribbling, but may be better than directly kicking it if there are opponents nearby.

In this running example, we illustrate the effects of different skills by using the mechanisms above in complementary ways to create opponent-aware ball-manipulation plans. First, we specify four macro-skills that the robot can take: align to shoot the ball, align to shoot using the dribbler, move the ball to a more beneficial location using the dribbler, and kick the ball. We define in detail the algorithms and physical limitations of these skills. Then, we use a skill decision algorithm to select among these skills depending on the state of the opponents.

We then provide evidence of the efficacy of our approach using two methods: statistics gathered

from the RoboCup 2015 competition, and in-lab experiments. The RoboCup statistics provide evidence of the effectiveness of the approach in real competitive games. To collect experimental data in a more controlled setup, we run repeated experiments of various soccer scenarios that illustrate the advantages of each of the defined skills. These experiments show that the various skills are successful in different scenarios, which supports the need for an opponent-aware decision process among the skills, as well as within each skill. Overall, this work highlights the larger picture that intra-robot replanning can improve team performance.

2.2 Problem Description

In the general robot soccer problem, each robot in the team must be able to effectively perform the individual skills selected by the centralized team planner. In this chapter's example, we assume that a robot ρ at location $\boldsymbol{l}_\rho$, currently in possession of the ball, must move the ball from its current location $\boldsymbol{l}_b$ to a target location $\boldsymbol{l}_t$, chosen by a separate team planner [Mendoza et al., 2016]. Thus, the robot needs to decide how to best move the ball to $\boldsymbol{l}_t$.

Our robots can manipulate the ball via two mechanisms: (i) a *kicker* enables the robots to impart momentum on the ball and thus perform shots or passes, and (ii) a *dribbler bar* enables the robots to impart backspin on the ball, and thus drive while maintaining contact with the ball. Kicking the ball enables the robots to move the ball quickly, but without protecting it. Dribbling the ball enables them to move the ball while guarding it; however, this method of moving the ball is slower, and it sometimes fails due to the robot losing control of the ball.

Due to the hardware design of the robots, the robot must have the ball immediately in front of it to be able to dribble it or kick it, i.e., the robot must face in direction $\boldsymbol{\phi} = (\boldsymbol{l}_b - \boldsymbol{l}_\rho)$, and be at a distance of approximately $r_\rho + r_b$ from the ball, where r_ρ and r_b are the radius of the robot and ball, respectively. Furthermore, the robots are only capable of kicking in the forward direction $\boldsymbol{\phi}$. Thus, to execute a pass or a shot, the robot must be facing both the ball and the target location $\boldsymbol{l}_t$.

To intelligently decide how to move the ball to $\boldsymbol{l}_t$, the robot must know (i) how to use its dribbler and kicker effectively, and (ii) how to evaluate the probability of success of different ways of using them. The use of the kicker and evaluating how likely it is for a pass or a shot to be successful has been researched previously [Biswas et al., 2014], and we use similar techniques in our example. The following chapter focuses on our approach to using the dribbling bar effectively and how to best choose among different dribbling and kicking skills.

2.3 Rationales: Possession and Alignment

In this section, we present the rationales used in our running example. Robot, ρ, has two conflicting objectives when in possession of the ball: aligning with the ball towards its target (*alignment*), and maintaining possession of the ball from the opponents (*possession*). These are the rationale of the current problem we are considering. In the previous CMDragon team, the focus was purely on *alignment* which we define as

$$\mathbb{A} = \frac{\boldsymbol{\phi}}{|\boldsymbol{\phi}|} \cdot \frac{(\boldsymbol{l}_t - \boldsymbol{l}_\rho)}{|\boldsymbol{l}_t - \boldsymbol{l}_\rho|} < \epsilon_a \wedge |\boldsymbol{l}_b - \boldsymbol{l}_\rho| < \epsilon_d \tag{2.1}$$

such that it is aligned with the target angle by less than $\cos^{-1}(\epsilon_a)$ and close enough to the ball within some ϵ_d [Biswas et al., 2015].

However, opponents introduce a threat that removes any guarantee on *possession*, and, by only considering *alignment*, the ball is often stolen. This failure to maintain possession can be attributed to two factors: the arc travel time of ρ and the opponent's proximity to the ball. Figure 2.3 demonstrates where the arc distance to *alignment* can take longer than the opponent's distance. In our simplified example, the opponent is very close in proximity to ρ and has an easy opportunity to gain *possession* of the ball by heading directly to it.

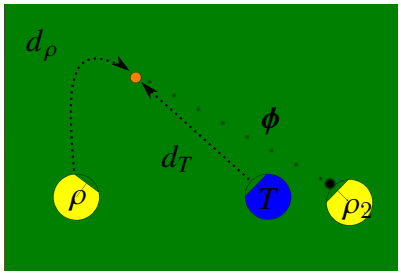

Figure 2.3: ρ drives to a position near the ball that aligns to pass to ρ_2 while T drives directly to take the ball.

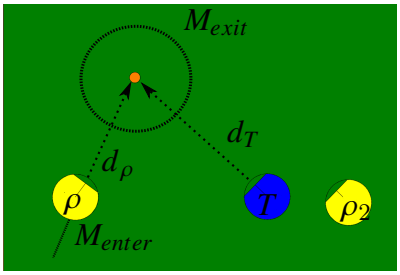

Figure 2.4: Variables used to determine if ρ should drive directly to the ball since T is threatening to take possession.

We define the objective of *possession* by describing d_ρ and d_T as our robot's and the closest opponent robot's distance to the ball respectively, M_{enter} as a proportional gain added to d_ρ, and M_{exit} as a constant distance from the ball. Shown in Figure 2.4, a *possession threat* ($\mathbb{P}$) is then defined to be true if

$$\mathbb{P} = d_\rho + (d_\rho * M_{enter}) > d_T \vee d_T < M_{exit} \tag{2.2}$$

Our approach maintains *possession* before considering *alignment*. If there is a *possession threat* then ρ drives directly to the ball and dribbles the ball. ρ is free to align itself if there is no threat.

These two rationales are constraining the action of kicking the ball to the new location. They are rationales used by the centralized planner in deciding the actions of the individual robot, and the centralized planner assumes that they remain valid during the execution of the plan. Considering the example again, we are providing these two specific rationale to the individual robot which is therefore acting as our rationale-driven plan in this example. In order for the rationale-driven plan to be useful, the individual robot needs to have methods that can fix the plan if a rationale becomes invalid during the execution of the plan. In the robot soccer domain, the rationale often become invalid because of the adversarial nature of the game as the opposing team constantly regains control of the ball through stealing or intercepting it.

2.4 Individual Robot Skills for Intra-Robot Replanning

This section covers the opponent-aware ball manipulation skills that are used for replanning. Considering *possession* and *alignment* as we defined them, we describe the Skill Decision Algorithm, which is an opponent-aware algorithm that implements the skills in an intelligent way that maintains ball possession. Lastly, we describe two dribbling skills used in the Skill Decision Algorithm.

2.4.1 Skill Decision Algorithm for Individual Skills

Based on the robot's manipulation mechanisms, we create four skills that can be used to move the ball to its target:

kick (K): Kicks the ball to $\boldsymbol{l}_t$. This is the quickest method of moving the ball to the target.

align-non-dribbling ($A_{\neg D}$): Aligns behind the ball by moving to the location $\boldsymbol{l}_b + \frac{(\boldsymbol{l}_b - \boldsymbol{l}_t) r_\rho}{|\boldsymbol{l}_b - \boldsymbol{l}_t|}$, where r_ρ is the radius of the robot. Shown in Figure 2.5.

dribbling-rotate (D_R): Dribbles the ball by approaching the closest location $\boldsymbol{l}_b + \frac{(\boldsymbol{l}_\rho - \boldsymbol{l}_b) r_\rho}{|\boldsymbol{l}_\rho - \boldsymbol{l}_b|}$ while facing the ball. It then quickly rotates to align to $\boldsymbol{l}_t$. Shown in Figure 2.6.

dribbling-move (D_M): Dribbles the ball by approaching it directly, and then moves the ball by pushing it toward $\boldsymbol{l}_t$, while avoiding obstacles. Shown in Figure 2.7.

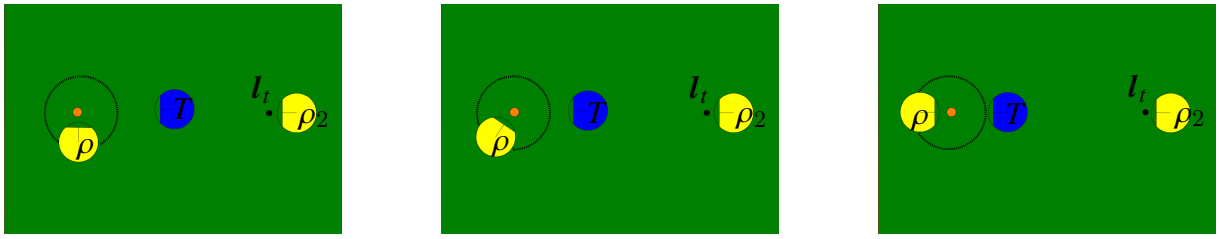

Figure 2.5: **Align-non-dribbling:** Drives around the ball (orange dot) on the dashed circle's perimeter to align to pass to target, $\boldsymbol{l}_t$ (black dot).

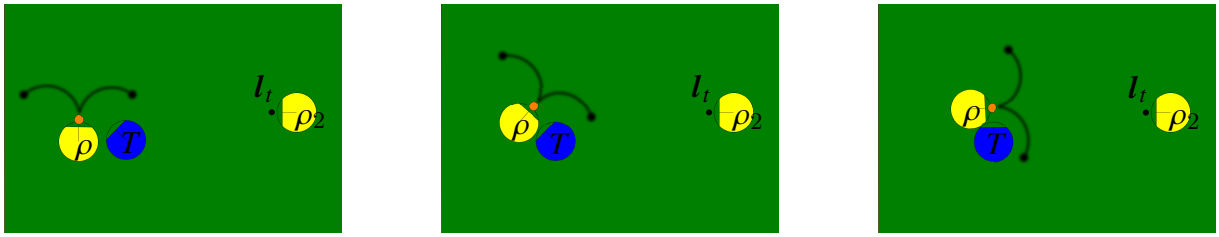

Figure 2.6: **Dribbling-rotate:** Dribbles the ball (orange dot), and pushes the ball along one of the black lines to align to pass to target, $\boldsymbol{l}_t$ (black dot).

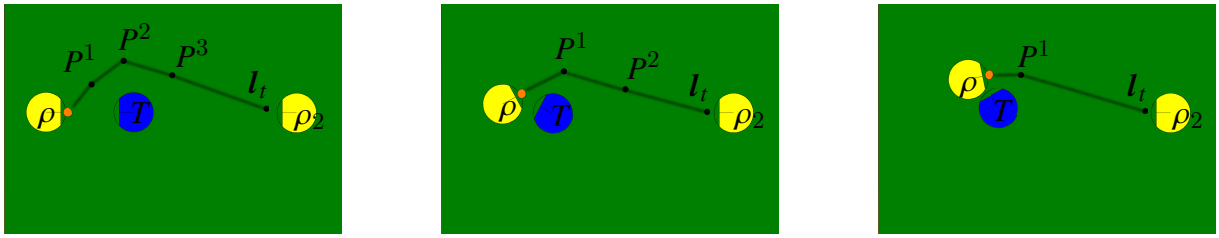

Figure 2.7: **Dribbling-move:** Dribbles the ball (orange dot) and pushes it along the path (black line) to $\boldsymbol{l}_t$ while avoiding obstacles such as the blue opposing robot, T.

Algorithm 2.1 Skill Decision Algorithm. **Inputs:** Team goal success probability, $E(\boldsymbol{l}_t)$, aligned with ball and target, $\mathbb{A}$, and possession threat, $\mathbb{P}$. **Output:** Skill.

```
1:  function SDA(E(l_t), 𝔸, ℙ)
2:      if E(l_t) <= δ then                  # Low success probability
3:          s = D_M                          # Move the ball to target location
4:      else                                 # Higher success probability
5:          if 𝔸 then                        # Checking alignment
6:              s = K                        # Kick if aligned
7:          else
8:              if ℙ then                    # Check if an opponent is near
9:                  s = D_R                  # Dribble the ball before an opponent steals it
10:             else
11:                 s = A_¬D                 # We have time to align nicely
12:             end if
13:         end if
14:     end if
15: return s                                 # Skill to execute
16: end function
```

The Skill Decision Algorithm (SDA), used by ρ at each time step, is shown in Algorithm 2.1. The team-goal's evaluation, $E(\boldsymbol{l}_t)$, is determined by multiple factors including: open angle, opponent interception time, and pass/shoot distance, which combine to create a probability of succeeding [Biswas et al., 2014]. Line 2 shows that if $E(\boldsymbol{l}_t)$ is less than or equal to a threshold δ, then the skill D_M is chosen to improve the current state, i.e., to improve the $E(\boldsymbol{l}_t)$. Otherwise, if $E(\boldsymbol{l}_t)$ is greater than δ then SDA checks alignment, $\mathbb{A}$, with $\boldsymbol{l}_t$, line 5. If $\mathbb{A}$ is true, then SDA kicks the ball to $\boldsymbol{l}_t$. Otherwise, SDA checks if there is a *possession threat*, $\mathbb{P}$, since aligning might lose the ball, line 8. If $\mathbb{P}$ is true, then SDA uses the skill D_R to grab the ball, protecting it, while still quickly aligning itself to kick to $\boldsymbol{l}_t$. Otherwise, SDA uses the more robust $A_{\neg D}$ skill to align itself around the ball. Lastly, the skill selected for execution is returned on line 15.

The task of the SDA is to execute a particular skill based on the current state of the rationales provided by the centralized planner. This implementation is limited to this particular domain, but the key elements of the algorithm can be generalized. First, SDA determines the rationales that are failing or are invalid, which happens through a if/then tree, or decision tree. Generalizing, the first task is to check all the rationales and determine which ones are currently invalid and store them in a list. Next, SDA determines what skill to execute by again relying on the tree structure to decide the skill. This can be generalized by determining which skills can fix certain invalid rationale and generating a new list of possible skills to execute to fix the plan. The list of skills would then need to be sorted based on some ordering, which would generalize the SDA's method of sorting using a tree structure. However, the new generalized ordering method could change based on the function used to sort the skills while the SDA's tree structure in this example is static. Lastly, the generalized method would return the first skill in the ordered list to be executed. Next, we discuss two of the skills used in SDA, which are representative of the replan policies used by the intra-robot replanning algorithm.

2.4.2 Dribbling-Rotate

D_R's priority is to align to $\boldsymbol{l}_t$ as quickly as possible, see previously described Figure 2.6. D_R dribbles the ball while maintaining an inward force to compensate for the centrifugal forces of the ball in order to maintain control as shown in Figure 2.8. It faces in the direction ϕ that provides the necessary centripetal force to maintain the ball on the dribbler of the robot: facing slightly inwards while turning provides a component of the normal force from the robot that always points towards the center of the circumference. Given the robot drives forward with speed s while gradually changing its orientation with speed ω, forming a circle of radius R, the constraint $s = \omega R$ holds in this case. The necessary angle offset ϕ can be obtained analytically by noticing that all the forces in Figure 2.8 need to cancel out in the rotating reference frame. Therefore, we obtain the pair of equations:

$$\begin{aligned} |\boldsymbol{f}_N| \sin\phi &= |\boldsymbol{f}_C| \\ |\boldsymbol{f}_N| \cos\phi &= |\boldsymbol{f}_F| \end{aligned} \tag{2.3}$$

Then, given the acceleration of gravity g, the coefficient of friction of the carpet under the robots μ, and the mass of the ball m (which cancels out in the end), we obtain:

$$\begin{aligned} |\boldsymbol{f}_N| \cos\phi &= m\omega^2 R \\ |\boldsymbol{f}_N| \sin\phi &= \mu m g \end{aligned} \tag{2.4}$$

Solving these equations for ϕ gives the result for the desired heading:

$$\phi = \tan^{-1}\left(\frac{\omega^2 R}{\mu g}\right) \tag{2.5}$$

We estimate μ by starting from measurements of when the robot kicks the ball. Then we locally optimize to the value that gives the best dribbling performance.

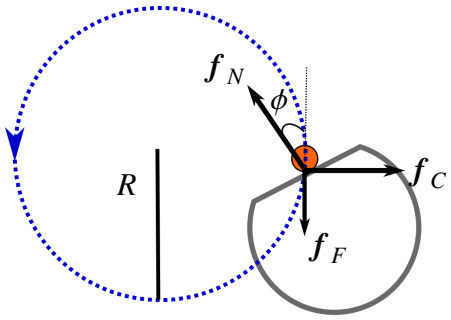

Figure 2.8: Robot dribbling the ball while facing slightly inwards. There exists an angle ϕ for which the forces are balanced.

2.4.3 Dribbling-Move

D_M's priority is to keep *possession* while driving towards $\boldsymbol{l}_t$ and avoiding all opponents and teammates, see previously described Figure 2.7. The priority of *alignment* naturally occurs as ρ drives

Algorithm 2.2 Dribbling-Move. **Inputs:** State of the world, W, robot, R, ball, B, and target, $\boldsymbol{l}_t$. **Output:** Location and angle.

function $Dribbling - Move(W, R, B, \boldsymbol{l}_t)$
$R_b = B_{loc} - R_{loc}$
$B_{front} = R_b.x > 0$
$B_{close} = R_b.x < MaxRobotRadius + (2 * BallRadius)$
$B_{ondribbler} = |R_b.y| \leq DribberWidth$
$B_{lost} = \neg(B_{front} \vee B_{close} \vee B_{ondribbler})$
$\{P^1, P^2, ..., \boldsymbol{l}_t\} = RRT(\boldsymbol{l}_t, W)$
$T = TurningThreat(W)$
    if B_{lost} ***then***
        $\{P^1, P^2, \ldots, \boldsymbol{l}_t\} = RRT(B, W)$
    else if $|R_{angle} - P^1_{angle}| > \alpha \vee |R_{loc} - P^1| < D_{min}$ ***then***
        $\theta = P^1_{angle}$
        if T ***then***
            $\theta = P^1_{angle} + 180°$
        end if
        return $\{R_{loc}, \theta\}$
    end if
$\theta = R_{angle} + (R_{angle} - P^1_{angle}) * \gamma$
return $\{P^1, \theta\}$
end function

toward $\boldsymbol{l}_t$ as shown in Figure 2.7. In Algorithm 2.2, D_M determines if ρ has the ball by checking: (i) if the ball is in front of ρ, B_{front}, (ii) if the ball is close to ρ, B_{close}, and (iii) if the ball is located somewhere on ρ's dribbler, $B_{ondribbler}$. If ρ loses the ball then D_M drives directly to the ball to regain possession. The path used to drive to $\boldsymbol{l}_t$ is generated by a Rapidly-exploring Random Tree (RRT) where the opponents and teammates are obstacles, as defined in [Lavalle et al., 2000]. The path is made of multiple intermediate locations, $(P^1, P^2, ..., P^{n-1}, \boldsymbol{l}_t)$. After any point P^n, D_M is always slightly turning ρ's forward direction towards the next point P^{n+1} by some empirically tuned γ. This maintains control of the ball while dribbling and moving. If the turning angle goes beyond a threshold, α, then D_M stops and rotates in place with the ball. α was empirically tuned by testing the limits of turning before the dribbler lost the ball ($\alpha = 40°$ for our experiments). If there is a *turning threat* such that an opponent, in close proximity, in the direction ρ is turning, then it turns in the opposite direction to protect the ball from being stolen [Mendoza et al., 2016]. D_M is complete once the ball arrives at $\boldsymbol{l}_t$.

2.4.4 Replan Policies

The ball-manipulation skills previously described are specific replan policies used to fix a certain rationale in the plan. They are very domain specific as they require some understanding of the possible failures of the rationales provided by the centralized planner and require a solution that would be feasible for the individual robot to execute within the current environment. In this thesis,

we describe many different replan policies for the different domains we describe in later chapters. They work in very similar ways to the skills we describe in this example. The replan policies take a set of inputs from the state and then execute a plan that attempts to fix one or more invalid rationales. Importantly, there may be replan policies that handle the same invalid rationale creating a choice for the individual robot (e.g., a robot not aligned to its desired pass location has a choice of D_M, D_R, or $A_{\neg D}$). In general, the replan policies attempt to fix the plan, by enabling the invalid rationale, and they have to comply with the other rationales, or constraints, provided by the centralized planner.

2.5 Experimental Evaluation

In this section, we detail multiple experimental evaluations of the SDA and the dribbling skills. These evaluations provide experimental evidence for the effectiveness and usefulness of individual robots having the ability to replan locally provided they have the rationale of the centralized planner. Similarly, after the introduction of the rationale-driven plan and the intra-robot replanning algorithm in this thesis, we provide experimental evaluations to demonstrate the generality and effectiveness of this approach in different domains, Chapters 5-7. These chapters, like the upcoming experiments and results in this section, provide experimental evidence and confirm the effectiveness of intra-robot replanning within a centralized controlled team.

2.5.1 RoboCup Results

In this subsection, we analyze the semi-final and the final game of the 2015 RoboCup Small-Size league, shown in the Table 2.1. We used the new skills (D_R, D_M) and SDA during these games in the tournament. The data was collected by analyzing the log files of the games. For our purposes, we define a successful pass as the ball reaching to its intended target, and a possession threat, $\mathbb{P}$, as described earlier (an opponent in close proximity that could steal the ball). In the tables below, the second column is the number of times a skill succeeded in passing the ball while the third column is the success rate of those passes actually reaching the teammate. This distinction is important since the skill might pass around the possession threat, $\mathbb{P}$, but the teammate fails or the ball is intercepted.

Semi-Final game		
Skill	**(Success/Total)**	**(Success/Total)**
First Part	**Total # of Uses**	**$\mathbb{P}$ + Pass**
$A_{\neg D}$	17/32	3/14
Second Part		
$A_{\neg D}$	9/28	1/6
D_R	11/15 ($\forall\mathbb{P}$)	4/11
D_M	10/17 ($\forall\mathbb{P}$)	7/10

Final game		
Skill	**(Success/Total)**	**(Success/Total)**
	Total # of Uses	**$\mathbb{P}$ + Pass**
$A_{\neg D}$	10/29	3/13
D_R	23/36($\forall\mathbb{P}$)	11/18
D_M	10/23 ($\forall\mathbb{P}$)	6/10

Table 2.1: Statistics for the three maneuvering skills during the Semi-Final and Final game at the 2015 RoboCup Small-Size League tournament.

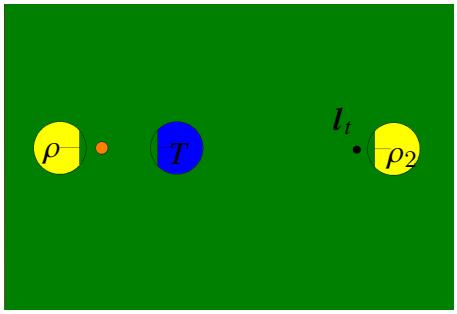

Figure 2.9: (EXP1) Used in the simulation evaluation, it is a passing with marking domain where T is facing off against ρ, who must kick to ρ_2.

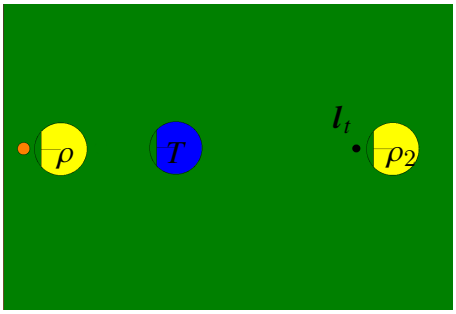

Figure 2.10: (EXP2) Used in the simulation evaluation, it is a passing with marking domain where ρ must get around the ball to align itself to kick to ρ_2.

The semi-final game is divided into two parts since for roughly half the game only $A_{\neg D}$ was used. In the first part, there were 14 passes with $\mathbb{P}$ and only 3 were successful using only $A_{\neg D}$. In the second part, we used SDA with the rule that dribbling was only allowed on the offensive side of the field. Therefore, on the defensive side, $A_{\neg D} + \mathbb{P}$ was used 6 times, succeeding 1 time. On the offensive side, D_R passed 11 times and succeeded 4 times with no clear improvement. D_M did improve with 7 successful passes out of 10 times. Interestingly, D_M was only used when under pressure by an opponent, which was the major cause of the low value of $E(\boldsymbol{l}_t)$ (< 0.1). Therefore, D_M started in a situation with a vastly low probability of success and under $\mathbb{P}$, but still it succeeded 7 times in getting away from the opponents and finding a better pass.

In the final game, we used SDA for the entire game with the same offensive restriction to dribbling. Again, we see poor performance for $A_{\neg D} + \mathbb{P}$ with 3 successful passes out of 13. D_R performed much better in this game with 11 successful passes out of 18, and D_M performed well again with 6 successful passes out of 10.

Real games only provide sparse amounts of information on the benefit of the added skills because they are short and unreproducible. Still, they provide evidence on the algorithm's performance in real-world conditions against unknown opponents for which they were designed to handle. The results show that $A_{\neg D}$ is very unsuccessful when there is a *possession threat*, and by implementing more intelligent ball-manipulation skills we improve the success rate against unknown opponents. Based on our review of the competition games, there were clear times when D_R was better than D_M and vice versa. To better understand our analysis of the game, we explore the *passing with marking* domain to challenge our robot with situations often found in robot soccer, specifically those with *possession threats*.

2.5.2 Passing with Marking

Passing with marking is a sub-domain of soccer that uses marking to induce a state where the probability of successfully passing is decreased due to the proximity of the opponent(s), i.e., a *possession threat*. The domain starts with one robot ρ being marked by a close opponent Taker, T, at some distance d_T. ρ is placed closest to the ball while T blocks the initial pass. As T's distance to the ball, d_T, decreases, it is more likely to gain possession or block the pass. The objective is for ρ to pass to ρ_2 before T steals the ball or kicks it out of bounds. We define stealing the ball as

when T has the ball within a robot radius plus a ball radius for at least 1 second. This constraint ends stalemates where both robots are driving into the ball and not moving. The domain is defined by a bounded area, and the teammate ρ_2 moves within this area to get open for the pass defined by its own team objective [Mendoza et al., 2016].

We devised two scenarios of the *passing with marking*: EXP1 where ρ is facing the ball and T, shown in Figure 2.9, and EXP2 where ρ faces away from T with the ball near its dribbler, shown in Figure 2.10. We ran both EXP1 and EXP2 in a physics-based simulation. For each test, we used only one of the approaches to see if the skill could pass the ball to ρ_2 using only that approach. We also devised two opponents that change the performance of the approaches. The *Drive to ball* opponent heads for the ball and tries to grab a hold of it. The *Clear ball* opponent attempts to kick the ball out of bounds, which usually involves it heading towards either the right or left side of ρ to kick it away.

EXP1 induces a state where T is blocking the initial pass and as d_T decreases the taker has a higher chance of stealing the ball away from ρ. This is clearly demonstrated in Table 2.2 where for both opponents the non-dribble approach $A_{\neg D}$ often fails to pass to its teammate. However, $A_{\neg D}$ does surprisingly better than D_R against the *Drive to ball* for two reasons. First, D_R fails at this task because as ρ approaches the ball so does T, and they often get stuck in a stalemate as D_R's forward velocity pushes against T. Second, $A_{\neg D}$'s success is due to luck as it kicks the ball immediately off of T and on occasion can get the rebound and pass to ρ_2. As d_T increases, the ball bounces less and $A_{\neg D}$ does not get as lucky, as shown in Table 2.2. D_M performs the best against *Drive to ball* since when it gets into the stalemate position it can sometimes rotate in place with the ball and move to a better passing position. Rotating in place allows D_M to succeed where D_R fails. Both D_R and D_M perform very well against *Clear ball* because the same stalemate position does not arise as often since T is trying to get to the side of the ball in order to kick it out of bounds. This provides these skills with the opportunity to dribble without getting stuck.

Physics-based Simulation EXP1 (Success/100)					
	Opponents				
Approach	**Drive to ball** ($d_T = \{260, 360, 460, 860\}$ mm)				**Clear ball** ($d_T = 260$ mm)
$A_{\neg D}$	15	11	2	1	3
D_R	4	2	0	14	89
D_M	35	48	53	97	84

Table 2.2: Passing with marking for 100 episodes on each approach. See Figure 2.9 for setup.

EXP2 induces a state where ρ has to get around the ball in order to align itself to pass to ρ_2, while T puts pressure from the back as it tries to steal the ball. In Table 2.3, $A_{\neg D}$ did the worst out of the three approaches, and its small success against *Drive to ball* is because T would sometimes get stuck behind ρ and remain behind it. D_R and D_M both did well against *Drive to ball*. D_M was the best because it simply rotated in place first to align itself and it did so in the opposite direction of T. This meant that T was often circling around on the backside of ρ, giving it a clear pass. However, rotating in place was D_M's downfall against *Clear ball* since T would maintain its heading until ρ rotated to a side and then T would kick it away. The reason D_R did the best against *Clear ball* was because it created more distance between itself and T as it circled around. For the same reason, it performed better as d_T increased while D_M remained relatively the same.

Physics-based Simulation EXP2 (Success/100)			
	Opponents		
Approach	**Drive to ball** ($d_T = 590\ mm$)	**Clear ball** ($d_T = \{590, 690\}\ mm$)	
$A_{\neg D}$	21	4	8
D_R	84	48	85
D_M	96	22	18

Table 2.3: Passing with marking for 100 episodes on each approach. See Figure 2.10 for setup.

We do not have a way of automating experiments on real-robots, and robots wear with use so it is not cost-effective to run hundreds of experiments on the real-robots. We did run the EXP1 with the *Clear ball* opponent on our real-robots. Each approach was tested 10 times, and the results in Table 2.4 are slightly different than our simulated experiment. $A_{\neg D}$ did very poorly as predicted by simulation, but the performances of D_R and D_M were not as expected. This difference can be attributed to factors on the complexity of executing skills in a stochastic environment with noisy actuators and perception, i.e., the simulation does perfectly describe the noise in wheel velocity, carpet slip, and vision locality error. D_M still performs well on passing to the teammate. D_R would often continuously circle while T blocked the pass and the ball was eventually stolen. An example run of $A_{\neg D}$ and D_M is shown in Figure 2.11.

Real-robot experiment (Success/10))	
	Opponents
Approach	**Clear ball** ($d_T = 355\ mm$)
$A_{\neg D}$	0
D_R	2
D_M	5

Table 2.4: Passing with marking using real-robots with 10 episodes each.

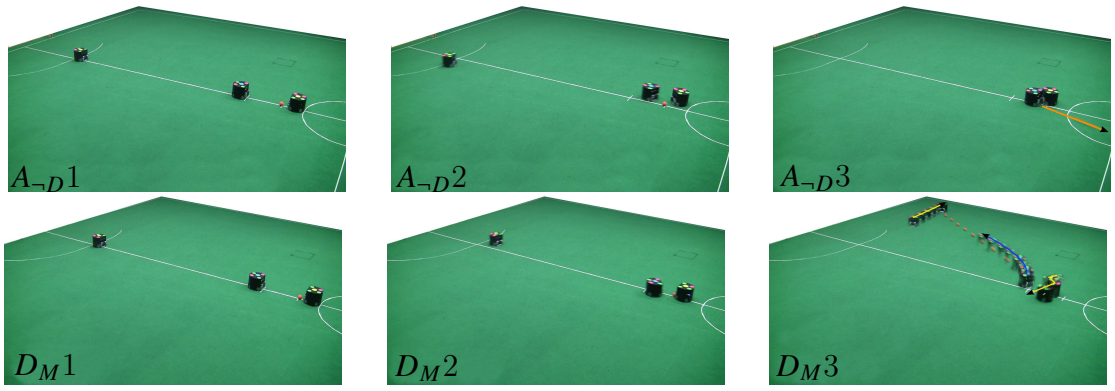

Figure 2.11: ($A_{\neg D}$): ρ rotates around the ball to pass but loses it as the opponent kicks it away. (D_M): ρ, using Algorithm 2.2, goes directly to the ball, slides past the opponent to the right, and passes to the teammate. ρ was 75 *mm* and T was 355 *mm* away from the ball in their initial positions. T was running *Clear ball*.

2.6 Summary

In this chapter, we demonstrate that an individual robot can improve the team performance by first considering the rationale of the actions that were chosen for it and then decide, if needed, to change the action to have a better success rate. This example demonstrates the key aspects of this thesis when considering the problem of intra-robot replanning. The following describes how the chapters of the thesis align with different aspects of the example that is described in this chapter.

Chapter 3 - In Section 2.3, we highlight two rationale used by the team planner when considering the action of passing to another team member, *possession* and *alignment*. These rationale were then used by the individual robot for decision making. In general, the individual robot should be provided with all of the reasoning for the plan provided by the centralized planner. We generalize the concept of rationale and we detail the process of extracting the rationale from the planner and adding it to the plan in Chapter 3.

Chapter 4 - In Section 2.4, we detail the individual action of the individual robots that are used for intra-robot replanning. Specifically, in section 2.4.1, the Skill Decision Algorithm details which skill is used based on the value of the rationale in the current state. This works for this specific example, however a more general concept is to have an algorithm that selects applicable skills, actions, or policies based on the value of rationales, provided in the plan, in the current environment. We detail the generalized intra-robot replanning algorithm in Chapter 4.

Chapter 5 - In Section 2.5, we describe different results in the robot soccer domain that demonstrate the benefits of our approach over the previous purely centralized method. We provide more results highlighting the benefits of intra-robot replanning in the robot soccer domain in Chapter 5.

Chapter 6 - We describe experiments within a different domain, specifically with autonomous underwater vehicles in a simulated ocean environment. In this domain, we highlight the benefits of intra-robot replanning when communicating with the centralized planner is costly. The ocean domain is in many ways different than our robot soccer domain as the ocean environment is very complex and hard to model. Therefore, experiments in the ocean domain highlight the generality of our approach.

Chapter 7 - We describe experiments in an abstract planning domain where a robot, in a simulated environment, must collect keys, unlock doors, and drop keys. This domain focuses on a more traditional planning problem and demonstrates the benefits of intra-robot replanning for a single robot communicating with a centralized planner that generates the rationale from its planning process while not having a complete model of the environment.

Chapter 8 - In our example, we show that different skills are useful in different situations that the soccer robot was involved in. In other words, a particular skill is more useful in certain states of the environment and less in others. This problem was not handled in our example because of the static nature of the Skill Decision Algorithm. In Chapter 8, we detail the process of learning a score for each replan policy, a skill in this chapter's example. The score is then used in our generalized intra-robot replanning algorithm to improve the performance even further.

Chapter 9 - We discuss previous work in areas relevant to the research of intra-robot replanning. These include plan representation, execution, failures, replanning, and learning.

Chapter 10 - We conclude the thesis with a summary of its contributions and the possible directions for future work.

Chapter 3

Rationale-Driven Plan

Planning does not include execution. Planning can be considered a sub-problem of the larger problem of planning and execution. Planning involves an initial state, a goal state, and a model of the agent(s) and the environment. A plan, a sequence of actions, is then generated using the model that goes from the initial state to the goal state for the agent(s). These plans can come in a variety of forms. Some examples are:

1. STRIPS: a sequence of actions with precondition(s) and effect(s) that take an initial state to a goal state through the effects of actions [Fikes and Nilsson, 1971].

2. A simple temporal network: a sequences of actions with start, duration, and end times for temporal flexible plans [Dechter et al., 1991].

3. A finite state machine: the current state (precondition) determines the next immediate action with the assumption that it will eventually lead to the goal state [Gill, 1962].

4. A Markov decision process (MDP) policy: a universal plan, a function that for any state produces an action [Bellman, 1957].

These all share a common feature that a model is used when generating the *plan*. However, the model may not equal reality. The model is an estimation, an approximation, of reality. Therefore, the model may be inaccurate and incomplete, so an *executing agent* may find that the generated plan does not match reality. MDPs attempt to handle this uncertainty, but generating MDPs for large problems can be very expensive and oftentimes excessive. Therefore, we consider linear plans, but ultimately the execution of such a plan fails when discrepancies between expectation and reality occur.

Going forward, we consider a *plan* as a sequence of actions with preconditions and effects (as defined by STRIPS). Provided such a plan, an *executing agent* can behave in a variety of ways when presented with a failure:

1. Indifferent: blindly continues executing actions without consideration for failures.

2. Re-plan: generates an entirely new plan from the failed state to the goal state.

3. Restore: attempts to fix or alter a subset of the old plan to handle the failed state.

Most research has focused on avoiding the indifferent executing agent as much as possible, and has primarily focused on re-plan because it essentially makes the problem of planning equal to planning and execution by continuously re-planning for all failures [Fritz and McIlraith, 2007]. The restore executing agent functions in the large spectrum between indifferent and re-plan. A majority of the previous research in these areas have assumed that planning and executing are accomplished by the same agent.

In this thesis, we presume the planning is performed by a centralized planner and the execution is performed by different agent(s). This division of labor is often seen in multi-robot domains where a centralized planner generates and sends the plan to each individual robot. We refer to the collective approach of re-plan and restore as *replanning*. When considering replanning, the simplest solution is to have the executing robot recognize a failure and then request a new plan from the centralized planner, allowing the centralized planner to decide on the type of replanning. However, this division creates a communication overhead between planning and executing, and state information needs to be exchanged and most likely simplified, i.e., the executing robot may know more about the world than the centralized planner. There can also be multiple executing robots that need to be informed about changes in the plan due to failures, leading to more overhead and considerations on how often the executing robot should be in contact with the centralized planner. Ultimately, the simplest solution can have detrimental effects on the performance of the executing robots(s).

Our alternative solution is to allow the executing robot to handle failures locally when possible, and defer to the centralized planner when there is no other option. The executing robot would then be categorized in restore as it will attempt to fix the previously provided plan before requesting a new plan. This theory was introduced as intra-agent replanning by [Talamadupula et al., 2013] where the individual executing agent must comply with the constraints of the other agents in the environment. In our case, the executing robot must comply with the centralized planner's rationale, i.e., its reasoning for why the actions and its parameters were selected into the plan, while replanning for failures. For our purposes, there are two vital concerns with the implementation of an intra-robot replanning robot: (i) the executing robot must detect failures, and (ii) the executing robot must restore its plan without being counter-productive towards the centralized planner's overall objective(s). There has been research in monitoring rationales in dynamic environments that occur during the planning phase, but it did not add all of the rationale to the plan or focus on rationales during the execution of the plan [Veloso et al., 1998].

This chapter introduces a rationale-driven plan which provides the reasoning of the centralized planner to the individual robot(s). In Section 3.1, we describe the rationale-driven plan and the different types of rationale contained within the rationale-driven plan. In Section 3.2, we describe the algorithms that generate the rationales that are added to the rationale-driven plan.

3.1 Elements of a Rationale-Driven Plan

The rationale-driven plan needs to store the rationale, or reasoning, of why the actions and their parameters were chosen by the centralized planner. Certain rationales can then be monitored by the autonomous robot in case they become invalid or change during execution, and then the rationales may be used, if needed, for replanning. We define the recognizable preconditions and effects of actions as the *structural rationale* of a plan, and they are often used in execution monitoring

for detecting immediate failures. However, we argue that the structural rationale only provide a fraction of the total reasoning within a planning system. During the process of creating or selecting actions, the planner makes many decisions about why actions are applicable and relevant to the plan. We consider these the *processing rationale* of the planner. Furthermore, the planner may locally or globally constrain the actions or state of the robot, in which case we refer to these as *constraint rationale*.

3.1.1 Structural Rationale

The primary method of detecting failures comes from the sub-goaling structure explored during the process of solving a planning problem. The preconditions and the effects of an action provide the conditions for an action and the resulting changes after the action is executed. They essentially provide the rationale behind why the actions are in their particular order. As previously stated, we refer to these as the structural rationale of the plan. Without these, the executing robot would not be able to check if it has failed an action (effects), or that an action is no longer possible (preconditions). The sub-goaling structure provides a clear method for monitoring execution for *immediate* failures:

- **Preconditions** provide information on what in the environment needs to remain valid before the action can be executed.
- **Effects** provide information on what the outcomes of the action are meant to change in the environment. With regard to failures, they are important for knowing if a goal or subgoal was achieved.

The structural rationale have been used numerously in research for execution monitoring, failure detection, and replanning in dynamic environments [Kambhampati, 1990].

Consider again the division of planning and execution. The centralized planner provides the structural rationale with the plan to the executing robots so that the executing robots can detect when an *immediate* failure occurs. The executing robot monitors its current state for the preconditions and effects of the action it is currently executing. Upon detecting a failure, the autonomous robot wants to restore the plan, if possible. However, we have assumed the autonomous robot is not the planner and therefore only contains the information provided in the plan. Assuming the autonomous robot has a method to restore the plan, then would the plan have provided enough information, given the actions and structural rationale, to restore the plan properly, i.e., in compliance with the centralized planner's objective(s)? Of course this depends on the planning algorithm, assumptions made by the planner, and in general, the rationale of the centralized planner. We argue the autonomous robot would generally not have enough information to replan properly. For example, there may be areas that are out of bounds for the robot, speed limits such that the robot should not go faster in an attempt to make up time, or a rather common assumption that the world should not change externally to the robot's actions.

3.1.2 Processing Rationale

In general, we claim there is a varying amount of information left out by the planner. Consider: i) the selection process of actions, why the actions were chosen in comparison with other possible

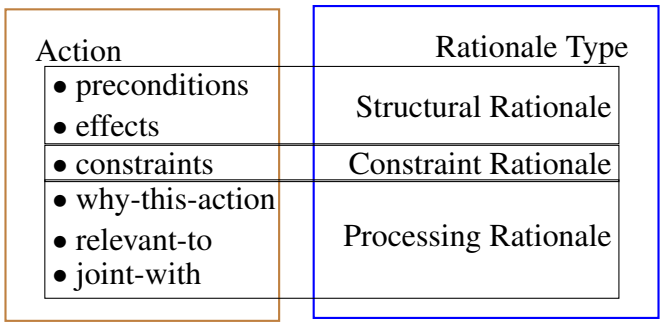

Figure 3.1: Rationales added to each action within a plan

actions; ii) the bounding of action parameters, why and how the values were determined for the parameters; and iii) the other robots involved in the plan. To facilitate intra-robot replanning, the executing robot needs to be provided with more information about the decision making of the centralized planner, i.e., include the planner's rationale.

This line of research follows a similar path to problem solving cases introduced by [Veloso and Carbonell, 1993]. As an approach to speed up planning, problem solving cases are stored with the reasoning and rationale of previous planning problems. The rationale are entirely derived from the planning process. This research focused on using the problem solving cases to speed up planning when solving a new problem. It did not consider replanning, adding this information to the plan, or the addition of multiple robots within the plan. That withstanding, it provides a reasonable starting point for considering rationale to add to the plan for intra-robot replanning.

Given an action in the plan, these are the questions that are of importance to the executing robot trying to replan locally:

- Why was this action added to the plan?
- Why was this particular action and its parameters chosen?
- What sub-goal(s) and goal(s) does this action contribute to achieving?
- What other robots are involved in this action?

As shown in Figure 3.1, we have new additions to the action that we have placed into different rationale types. The processing rationale captures the information on why this action was added to the plan, why this action and its parameters were chosen, why was it needed in the plan, i.e., for what sub-goal and goal, and what other robots are involved in the action.

Why-this-action: is a set of functions with their associated parameters. They provide reasoning on why an action and its parameters were chosen for the plan. Examples are:

- *select* with associated parameters for selection reason.
- *prefer* with the preferred name and the alternative actions that were not preferred.
- *reject* with the rejected name and the alternative actions that were rejected leading to the selected action.
- *why-user* with a function name provided by a user and its associated input parameters.

Relevant-to: refers to the goal that needs this action or condition to be achieved.

Joint-with: provides a set of references to the other robots involved in the action. This can be a set of robot IDs or be empty for an action executed alone. This information is very important in informing the robot that the action is a joint-action [Levesque et al., 1990]. In particular, the robot needs to consider the other robots when it is attempting to replan a joint action.

3.1.3 Constraint Rationale

We define the constraint rationale as the facts that are checked or assumed by the centralized planner but there is no explicit plan to achieve them, unlike the preconditions that the centralized planner may try to achieve, i.e, to plan for. The constraint rationale are included in the domain by human operators and/or by the planner's model. They also may refer to coordination requirements with other robots in the plan. We represent a constraint rationale as a function with a finite set of parameters (p_i):

$$f(p_1, p_2, ..., p_n) \rightarrow \{true, false\} \tag{3.1}$$

that returns true or false given those specific parameters. An example constraint is that the "sky is clear" for the current plan of "walk to store" to remain valid by the planner's rationale, because if the sky was not clear, then the plan would need to be "walk to store with umbrella." We could represent this pseudo constraint as the following function:

$$\begin{aligned} \texttt{SkyClear}(cloudcoverage, precipitation) = cloudcoverage < 50\% \\ \wedge\, precipitation \equiv 0 \end{aligned} \tag{3.2}$$

where the sky is only clear if the cloud coverage is less than fifty percent and there is no detectable precipitation.

3.2 Generating a Rationale-Driven Plan

We have defined the information necessary for our rationale-driven plan in the previous section, however, generating this information is very planner dependent and the rationales vary between planners. Therefore, in this thesis, we detail select parts of a classical planner that extracts the rationale information, with the assumption that other planners can adapt these parts as needed. We assume a planning problem $P = \langle A, C, I, G \rangle$ where A are the actions, C are the constraints, I is the initial state, and G are the goals. We assume A has preconditions, effects, and parameters.

Algorithm 3.1 adds the rationales when the planner is deciding if an action is applicable given a state. As the planner loops over the actions, line 1, the planner must pick parameter values for the action if needed. The rationale for picking these parameters is stored in line 2. For example, the parameter for an action "move" may be the speed, and the speed limit may be one of multiple rationales used to pick the value for the speed. The rationales are created by the developers of the planner and are assumed to be shared with the individual robots. The preconditions and effects are added in lines 3 and 4, respectively. The action is then checked to be applicable in the state, and the constraints are checked to be true given the action in line 5. Then the constraints are added in line 6.

Algorithm 3.1 Returns applicable actions, $\hat{\mathcal{A}}$, provided state, $\mathcal{S}$, actions, $\mathcal{A}$, and constraints, C.

1: **for all** $a \in \mathcal{A}$ **do**
2: $a.\textit{why-this-action} \leftarrow \textit{pick-parameters}(a)$
3: $a.\textit{precondition} \leftarrow \textit{preconditions}(a)$
4: $a.\textit{effects} \leftarrow \textit{effects}(a)$
5: **if** $a.\textit{precondition} \in \mathcal{S} \wedge \{\forall C(a, \mathcal{S}) = \textit{true}\}$ **then**
6: $a.\textit{constraints} \leftarrow C$
7: $\hat{\mathcal{A}}$.insert(a)
8: **end if**
9: **end for**
10: **return** $\hat{\mathcal{A}}$

Robot	**Defender** (D_1)
Action	KICK BALL
parameters	$\langle \theta, X, Y, B_x, B_y, B'_x, B'_y, \rangle$
precond.	LOC(D_1, $X = 3$, $Y = -0.8$)
	HEADING(D_1, $\theta = 45°$)
	LOC(B, $B_x = 3.5$, $B_y = -0.8$)
effects	$\neg$LOC(B, $B_x = 3.5$, $B_y = -0.8$)
	LOC(B, $B'_x = 5.6$, $B'_y = 2.1$)
constraints	*not-facing-our-goal-side*
why-this	*pass-to-*D_2
-action	*pass-open*
	current-loc

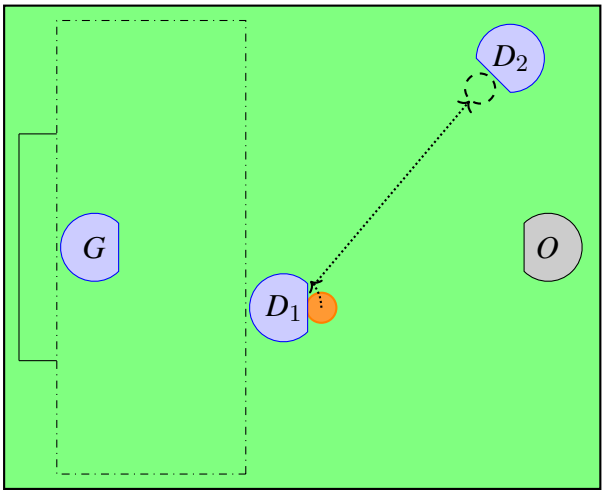

Figure 3.2: Example to help illustrate Algorithm 3.1. The illustration shows D_1 plan to kick the ball (orange dot) to D_2 (at the dashed circle location). The KICK BALL action for D_1 shows the rationale provided by the Algorithm 3.1.

Consider an example in the robot soccer domain in Figure 3.2 to help illustrate Algorithm 3.1 (see Chapters 2 & 5 for more details on the robot soccer domain). The centralized planner is checking applicable actions for the robot D_1 (Defender 1). Here, the action KICK BALL is being checked to see if it is applicable. On line 2, the function *pick-parameters* returns the rationales for why these parameters in the action were chosen. The θ parameter was chosen because the pass is open (*pass-open*) in that direction. The X, Y were picked simply because that is the current location of D_1 (*current-loc*). If the precondition X, Y parameters become invalid then the rationale *current-loc* informs D_1 on how to fix the values. The remaining parameters involve the ball's (B) location. They were decided on in order to pass the ball from D_1 to its teammate D_2, (*pass-to-*D_2). Now D_1 is informed that it is kicking the ball to the new location in order to pass the ball to its teammate. On lines 3 & 4, the preconditions and effects of the action are added. On line 5, assume the preconditions are valid in the state ($\mathcal{S}$). Next, the constraints are tested. For this example, assume that the users of the robot soccer team do not want the robot to kick the ball backwards towards its own goal. In this case, the angle of the robot is checked and determined to be valid,

so the constraint, *not-facing-our-goal-side*, is added to the action. In contrast, consider that the next action is KICK BALL except with an angle of $145°$, i.e., facing to pass to the goalie, G. The constraint would fail and the action would be deemed not applicable. This process would continue as the centralized planner checked the applicability of actions during the search for the final plan.

Algorithm 3.2 selects one of the applicable actions to continue the search and provides the rationale for why this particular action was chosen from the applicable ones. An action is selected in line 1. The rationale, or reason, for selecting this particular action is then stored in line 2.

Algorithm 3.2 Selecting an action given current state, $\mathcal{S}$, and applicable actions, $\hat{\mathcal{A}}$

1: a =*select-action*$(\mathcal{S}, \hat{\mathcal{A}})$
2: a.*why-this-action* $\leftarrow$ *used-selection-rule*

Robot	**Attacker** (A_1)
Action	KICK BALL (A_2)

Action
why-this-action
	KICK BALL (Goal Shot)
	prefer-ordering {
	goal → kick → dribble
	}

Action	DRIBBLE BALL

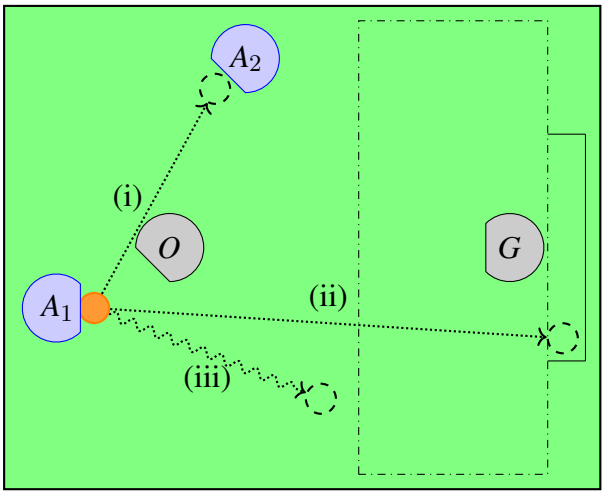

Figure 3.3: Example to help illustrate Algorithm 3.2. The illustration shows the applicable actions that the centralized planner has for A_1 currently in its planning process. These actions are: (i) Kick the ball (orange dot) to A_2 (at the dashed circle location), (ii) Kick the ball into the goal, and (iii) Dribble the ball (squiggly line) to a new better location. The action KICK BALL has been selected and the rationale is stored in *why-this-action*.

Consider an example in the robot soccer domain in Figure 3.3 to help illustrate Algorithm 3.2. The current applicable actions are (i) to kick the ball to teammate, A_2, (ii) to kick the ball into the goal, and (iii) to dribble the ball to a new location (squiggly line in figure). The centralized planner must make a decision regarding which of the actions it continues the search with. If the planner chooses at random, the rationale could be *randomly-selected*. However, in our example, we have a preferred ordering for the applicable actions, i.e., goal shots are preferred, then passes, then dribbling the ball. Therefore, the KICK BALL (Goal Shot) is selected and provided the rationale *prefer-ordering*.

Algorithm 3.3 adds rationales to each action once the action is in the final plan. Line 1 adds the goal that action a is used to achieve. Line 2 adds the robots that are tasked to execute this action with the robot given this action, $\hat{r}$. Line 3 loops over all of the effects of the action. Line 4 stores the rationale of why the effect is needed in the plan. This may be required as a precondition

of one of $\hat{r}$'s actions or for some robot in the team, in which case it would include the robot(s) as part of the rationale. However, the effect may not be required in the plan, a superfluous effect of the action. In that case, the rationale would be labeled superfluous so that $\hat{r}$ would know to ignore it during intra-robot replanning. Lastly, the planner may have more constraints, given the entire finalized plan, that the robot needs to maintain during execution, line 6. For example, certain predicates previously achieved and not needed as a precondition of this action need to remain for an action in the future, i.e., persistent conditions similarly defined in [Kambhampati and Hendler, 1992].

Algorithm 3.3 Rationales added during final construction of the plan path, given an action, a, current state, $\mathcal{S}$, goal, g, relevant to a, the robot executing this action, $\hat{r}$, all the robots, $\mathcal{R}$, involved in the plan, and any new constraints in the final plan, C.

1: $a.$*relevant-to* $\leftarrow g$
2: $a.$*joint-with* $\leftarrow r \subseteq \{\mathcal{R} - \{\hat{r}\} \lfloor joint(r, a)\}$
3: **for all** $e \in a.$*effects* **do**
4: $\quad e.$*rationale* $\leftarrow$ *why-needed*$(e, \mathcal{R})$
5: **end for**
6: $a.$*constraints* $\leftarrow C$

Consider an example in the robot soccer domain in Figure 3.4 to help illustrate Algorithm 3.3. In this example, we have a plan for two robots to score a goal. For simplicity, we have only added the rationales that are relevant to the example (the effect rationale relevant to preconditions of future actions have been removed for simplicity). The first action of A_1 is to dribble and move the ball to the new location. The *relevant-to* rationale relates to the sub-goal of positioning the ball at location (B_x, B_y). This action is not a joint action and therefore *joint-with* is null. The constraint *has-ball* is the assumption made by the centralized planner that the robot maintains control of the ball the entire time it dribbles the ball to the new location. The next action for A_1 is to rotate its position. The *relevant-to* is the needed precondition of the heading for the next KICK BALL action. Again, this is not a joint action. The constraint is that while A_1 is rotating there should not be any possession threat, i.e., there is not an opposing robot close enough to steal the ball. The final action for A_1 is to kick the ball to its teammate A_2. The *relevant-to* is the sub-goal of placing the ball at the location (B'_x, B'_y). This action is a joint action with A_2 as they must work together to maintain possession of the ball. The rationale for the effect of the new ball location is the precondition of the RECEIVE action of A_2. The constraint is that the path of the kick must remain open for the action to be valid. The first action of A_2 is to position itself at the new location (X, Y). This is not a joint-action and there are not extra constraints made by the centralized planner. The next action is for A_2 to receive the ball at its new location. This action is a joint action with A_1, as previously stated. The constraint added is the assumption that the path is open so that A_2 can successfully receive the ball. The final action of A_2 is to kick the ball into the goal. The *relevant-to* is the goal which is to score a goal, or get the ball into the goal box. This is not a joint action and the constraint is that the path must be open for the kick to be successful.

Robot	**Attacker** (A_1)
Action	DRIBBLE BALL
relevant-to	LOC(B, B_x, B_y)
joint-with	$\emptyset$
constraint	*has-ball*
Action	ROTATE
relevant-to	HEADING(A_1, θ)
joint-with	$\emptyset$
constraints	$\neg$*possession-threat*
Action	KICK BALL (A_2)
relevant-to	LOC(B, B'_x, B'_y)
joint-with	A_2
effects	*Loc*(B) $\rightarrow$ A_2, RECEIVE
constraint	*path-open*

Robot	**Attacker** (A_2)
Action	POSITION
relevant-to	LOC(A_2, X, Y)
joint-with	$\emptyset$
Action	RECEIVE
relevant-to	LOC(B, B'_x, B'_y)
joint-with	A_1
constraint	*path-open*
Action	KICK BALL (Goal Shot)
relevant-to	SCORE-GOAL
joint-with	$\emptyset$
constraint	*path-open*

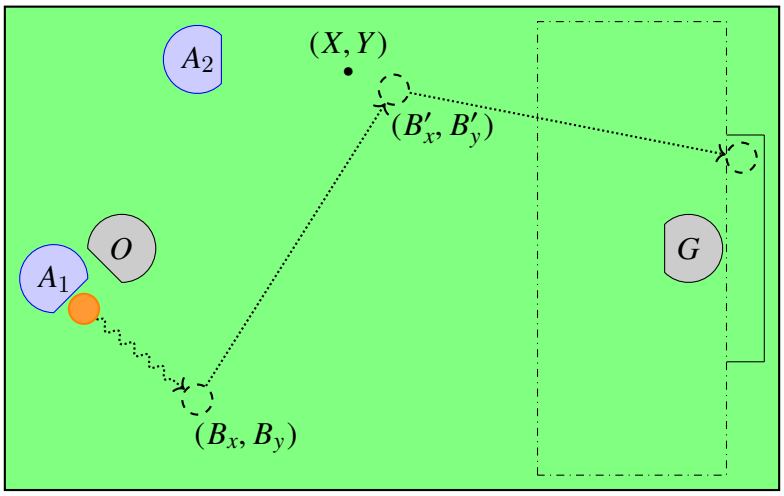

Figure 3.4: Example to help illustrate Algorithm 3.3. The plan for A_1 is: to dribble the ball, then rotate, then kick the ball to A_2. The plan for A_2 is: to position itself at new location (X, Y), then receive the ball, then kick the ball into the goal. The relevant rationales have been added to each action in their respective plans.

3.3 Summary

In this chapter, we present algorithms that are the building blocks of a typical action-based planner, and we demonstrate how the rationale for the rationale-driven plan can be extracted during these critical parts of the planning process. Every planner is different and the exact method for getting the rationale differ. Depending on the search method, some rationale information may not be explored by the planner and therefore not added. The key concept of our rationale-driven plan is to add the rationales that the centralized planner uses while checking applicable actions, selecting an action to continue the search, and constructing the final plan, so that the rationales may be passed along to the individual executing robot.

Given the rationale-driven plan, the individual robot is now better informed on the rationales of why the actions and their parameters were chosen in the plan. The next major task of the individual robot is use the rationale-driven plan to replan proactively and successfully. The individual robot needs to monitor the rationale of the plan and then, if there is a failure, decide how to replan locally using the rationale-driven plan. These questions are tackled in the next chapter.

Chapter 4

Intra-Robot Replanning

In this chapter, we present our method for intra-robot replanning, i.e., how an individual robot, provided a rationale-driven plan from a centralized planner, handles a failure locally during execution while remaining compliant with the centralized planner. We first formally define the problem of intra-robot replanning to better understand the core concept of the problem. Next, we describe the intra-robot replanning algorithm that we develop to tackle the intra-robot replanning problem. Following the description, we provide a simplified example that walks through the algorithm to help explain each part in concrete terms. Later chapters explore different domains that include experimental results to justify the importance and benefits of our intra-robot replanning algorithm with the rationale-driven plan. In this chapter, we only provide a simplified example to illustrate the algorithm.

4.1 Problem Definition

We describe the intra-robot replanning problem as the following tuple $\langle \mathcal{R}, \mathcal{S}, \mathcal{P}, P, \mathcal{T}, \mathcal{F} \rangle$:

- $\mathcal{R}$ is the set of robots involved in the plan;
- $\mathcal{S}$ is the current state;
- $\mathcal{P}$ is the set of replan policies; where a policy $\mathcal{P}_j = \langle \alpha, \beta, C \rangle$:
  - α are the input parameters to the replan policy,
  - β are the rationales that can be enabled by this policy (effects),
  - C are the constraints that the policy abides by during execution;
- P is the rationale-driven plan;
- $\mathcal{T}$ is a set of rationales for the current action;
- $\mathcal{F} \subseteq \mathcal{T}$ are the rationales that are invalid.

We assume the individual robot has a set of replan policies, $\mathcal{P}$, that can fix a set of invalid rationale. We always assume there exists at least one replan policy that calls the centralized planner

for a new plan; therefore, at least one replan policy exists that can handle all failure cases and all invalid rationale. In other words, the individual robot calls the centralized planner if there is no replan policy that can specifically handle a given invalid rationale that the robot is currently attempting to fix. We also assume that a replan policy does not invalidate rationales that have already been fixed by a previous replan policy.

There can be multiple replan policies that handle a single rationale. Therefore, when a rationale becomes invalid, the individual robot can have multiple different policies that can potentially re-enable the rationale. This leaves the individual with the task of picking a replan policy. We also want the individual robot to pick the best replan policy (based on some known metric), and so, we discuss the issue of sorting the replan policies to improve the performance of the intra-robot replanning [Cooksey and Veloso, 2017, Cooksey and Veloso, 2018]. Moreover, there is another concern to be addressed on sorting the rationales that are failing. First, we describe the intra-robot replanning algorithm.

4.2 Intra-Robot Replanning Algorithm

Algorithm 4.1 Intra-Robot Replanning algorithm for selecting failing rationales of action a, sorting the rationales, selecting applicable replan policies, sorting the replan policies, and then executing the policies until the rationales are true.

Require: $a, P, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathbb{P} = \{\emptyset\}$

1: $\mathcal{F} \leftarrow \{c | c \in a.constraints \wedge c \neq true\}$ # Find all invalid *constraints*
2: $\mathcal{F} \leftarrow \{c | c \in a.why\text{-}this\text{-}action \wedge c \neq true\}$ # Find all invalid *why-this-action*
3: **if** a is not executed **then** # Has the action not been executed yet?
4: $\mathcal{F} \leftarrow \{c | c \in a.preconditions \wedge c \notin \mathcal{S}\}$ # Then find all invalid *preconditions*
5: **else**
6: $\mathcal{F} \leftarrow \{c | c \in a.effects \wedge c \notin \mathcal{S}\}$ # Else find all invalid *effects*
7: **end if**
8: $\mathcal{F} \leftarrow$sort($\mathcal{F}$) # Sort invalid rationales
9: **for all** $\mathcal{F}_i \in \mathcal{F}$ **do** # Loop over all invalid rationales
10: **for all** $\mathcal{P}_i \in \mathcal{P}$ **do** # Find applicable replan policies
11: **if** $\mathcal{F}_i \subseteq \beta \in \mathcal{P}_i \wedge a.constraints \subseteq C \in \mathcal{P}_i$ **then**
12: $\mathbb{P}$.push($\mathcal{P}_i$) # Enables rationale and follows constraints
13: **end if**
14: **end for**
15: $\mathbb{P} \leftarrow$ sort($\mathbb{P}$) # Sort applicable replan policies
16: **repeat**
17: execute($\mathbb{P}$.front(α)) # Execute replan policy ($\alpha \subseteq a \cap P \cap \mathcal{S} \cap \mathcal{R}$)
18: $\mathbb{P}$.pop()
19: **until** $\mathcal{F}_i(p_1, ..., p_n) = true$ # End loop once rationale is valid
20: **end for**

Algorithm 4.1 is the intra-robot replanning algorithm that determines the invalid rationales, stored in $\mathcal{F}$, and determines the replan policies to fix the rationales. The algorithm requires the

current action, a, the current state, $\mathcal{S}$, the robots, $\mathcal{R}$, and the replan policies, $\mathcal{P}$. On line 1, all the constraints of the action are tested and any invalid constraints are added to $\mathcal{F}$. On line 2, the reasons behind why this action was chosen are checked, and if any are no longer valid, they are added to the list of invalid rationale, $\mathcal{F}$. If the action has not yet been executed, line 3, then the preconditions of the action are checked and invalid ones are added, line 4. Otherwise, the effects are checked and invalid ones are added, line 6. The rationale are then sorted by some cost function or known ordering, line 8. The next part of the algorithm determines the replan policies for the invalid rationale.

For every invalid rationale (line 9), the applicable replan policies $\mathbb{P}$ are selected by checking that the replan policy can enable the invalid rationale as provided in β of $\mathcal{P}_j$ in the problem definition and that the replan policy can handle the known constraints of the current action (lines 10-11). If $\mathcal{P}_j$ has the invalid rationale and can handle the constraints, then the replan policy is placed into the $\mathbb{P}$ queue (line 12). In line 15, the replan policies are sorted provided there is some cost function or predetermined ordering for the replan policies. While the rationale remains invalid, the replan policies in $\mathbb{P}$ are executed and popped (lines 16-19). The replan policy is provided the failed action, the failed plan, the current state, and the robots of the team. The replan policies are then tasked to use the rationale provided by the centralized planner and repair the plan, if possible. As previously stated, the assumed replan policy that calls the centralized planner ultimately ends the loop, if needed. Assuming that sorting has time complexity $n\log(n)$ where n is the size of the array being sorted, our algorithm has a running time complexity of $\mathcal{F}(\log(\mathcal{F}) + \mathcal{P}\log(\mathcal{P}))$.

4.2.1 Illustrated Example

In this example, we use a simplified robot soccer domain to illustrate the intra-robot replanning algorithm. We assume the rationale-driven plan has already been generated and provided to the robots. Specifically, we look at a particular point in time during the execution of one action for each robot. For clarity, we only add a sub-set of all the rationales that would normally be added to the actions and as such only the rationales relevant to our example have been added.

See Table 4.1 for the actions and rationales for the defense robots (we have excluded the goalie's plan for simplicity). The *relevant-to* rationale for both defending robots is **block-goal**, i.e., stop all possible incoming shots from opponents. The two defenders are executing a joint action with different preconditions, effects, and the joint-with rationale. For an illustration of our example scenario see Figure 4.1. The goalie is blocking the direct shot on goal, and the two defenders are holding positions that block non-direct shots on the goal. They are holding a "gap defense" such that if a shot is taken between them, then they can quickly close the gap and block the shot before it enters the defense zone (the dashed rectangle in the figure). Another rationale provided to the defenders is *ball-far* which means if the ball gets too close then the gap defense action is no longer valid. The constraint both defenders have is *not-in-defense-zone* because the defenders' positions are restricted to be outside the defense zone.

Replan Policies

For our example, we describe a few replan policies that each individual robot has in case different rationales become invalid during the execution of their plan.

1. EXIT-DEFENSE-ZONE $\langle \beta$ = *not-in-defense-zone*, $\forall C \rangle$: Takes shortest route to get outside of the defense zone, so the robot exits to the nearest side of the rectangular defense zone.

2. SINGLE-GAP-DEFENSE $\langle \beta$ = *joint-block*, $\forall C \rangle$: Blocks the direct heading of the opposing robot (with the ball) towards the goal, but it does not go beyond its original gap defense location and its teammates gap defense location.

3. MOVE-TO-LOC $\langle \beta$ = LOC(...), $\forall C \rangle$: The robot moves to the location provided by the location rationale, while abiding by the constraints provided by the centralized planner.

4. CENTRALIZED $\langle \forall \beta, \forall C \rangle$: Continues current action and waits for centralized planner to provide new plan.

Failure Scenario

Consider the scenario shown in Figure 4.2. D_1 is positioned correctly to defend the goal from its current location. Conversely, D_2 is far from its correct position, which means the joint action GAP-DEFEND is failing. For this situation, we discuss how the two defenders handle this failure using the intra-robot replanning algorithm along with the previously described replan policies. We assume for this example that the action, GAP-DEFEND, is being executed and therefore the *effects* are being checked.

Defender 1 (D_1): First the intra-robot replanning algorithm checks the rationales provided with the action (rationales marked with ¬ are invalid).

- *constraints*: *not-in-defense-zone*
- *why-this-action*: ¬*joint-block*, *ball-far*, *pre-current-loc*, *pre-current-angle*
- *effects*: LOC(D_1, X_1', Y_1'), ANGLE(D_1, θ_1')

Therefore, our set of invalid rationales is:

$\mathcal{F}$ = (¬*joint-block*)

For sorting the rationales, we only have one invalid rationale so nothing is sorted. The *joint-block* rationale is invalid because its teammate, D_2, is not in the correct position to block with D_1. The algorithm then loops over the four replan policies and finds SINGLE-GAP-DEFENSE and CENTRALIZED can handle the invalid rationale and the constraints provided in the action.

Next, the replan policies are sorted. This sorting could cause vastly different outcomes for D_1. If CENTRALIZED was sorted to be first, then D_1 would do nothing more than wait for the centralized planner to issue a new plan, while at the same time continuing to block its side of the gap defense. This would be equivalent to not using intra-robot replanning and therefore not particularly interesting. So for this example, SINGLE-GAP-DEFENSE is ordered first and therefore executed by our intra-robot replanning algorithm. There is no way for D_1 to explicitly re-enable *joint-block* because it is a rationale involving another robot that is not in its control. Therefore, SINGLE-GAP-DEFENSE attempts to block the direct angle of the opposing robot and thereby maintain the objective of *joint-block* which is the goal of blocking goal shots, **block-goal**. Once the teammate, D_2, gets close enough to the correct position then SINGLE-GAP-DEFENSE repositions D_1 into its

previous position and the replanning loop ends because the rationale *joint-block* is valid. The alternative is that the centralized planner provides a new plan while D_1 is replanning, and this forces D_1 to execute the new plan instead. It is important to note that without the rationale *joint-block*, D_1 would never have known it was failing the joint action.

Table 4.1: Part of a team plan provided to the defense of a robot soccer team with rationales. For simplicity, we only add rationales relevant to our example and exclude the numerous constraints that would normally be in this plan.

Robot	**Defender 1** (D_1)	**Defender 2** (D_2)
Action	GAP-DEFEND	GAP-DEFEND
preconditions	LOC(D_1,X_1, Y_1)	LOC(D_2,X_2, Y_2)
	ANGLE(D_1, θ_1)	ANGLE(D_2, θ_2)
effects	LOC(D_1, X_1', Y_1')	LOC(D_2, X_2', Y_2')
	ANGLE(D_1, θ_1')	ANGLE(D_2, θ_2')
constraints	*not-in-defense-zone*	*not-in-defense-zone*
why-this-action	*joint-block*	*joint-block*
	ball-far	*ball-far*
	pre-current-loc	*pre-current-loc*
	pre-current-angle	*pre-current-angle*
relevant-to	**block-goal**	**block-goal**
joint-with	D_2	D_1

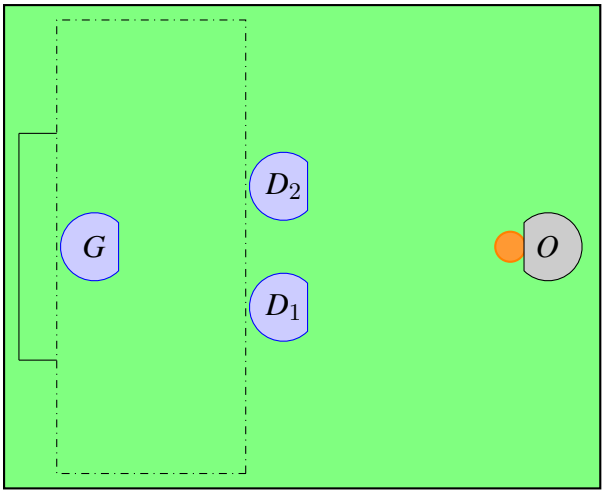

Figure 4.1: The two defenders (D_1 & D_2) are aligned with the goalie (G) to block shots from the opposing robot, O, who is in control of the ball (orange circle).

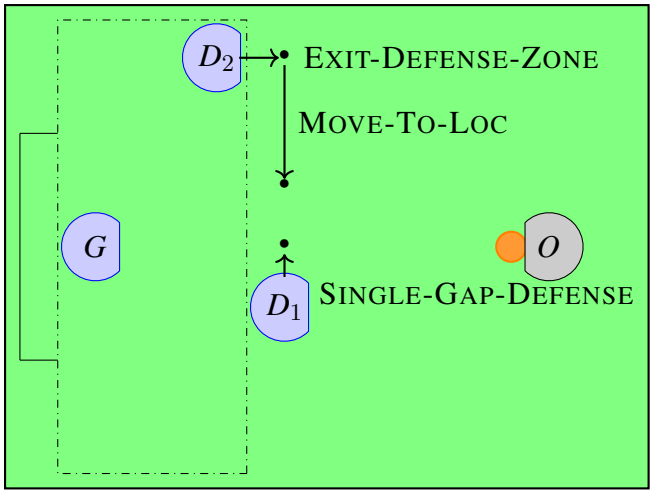

Figure 4.2: The defender D_1 is positioned correctly, however its teammate D_2 is not positioned to defend the goal. The lines show the robots' movements along with the replan policies that are handling a particular failure.

Defender 2 (D_2): First the intra-robot replanning algorithm checks the rationales provided with the action (rationales marked with ¬ are invalid).

- *constraints*: ¬*not-in-defense-zone*
- *why-this-action*: ¬*joint-block*, *ball-far*, *pre-current-loc*, *pre-current-angle*
- *effects*: ¬LOC(D_1, X_1', Y_1'), ANGLE(D_1, θ_1')

Therefore, our set of invalid rationales is:

$\mathcal{F}$ = (¬*not-in-defense-zone*, ¬*joint-block*, ¬LOC(D_1, X_1', Y_1'))

The next step of sorting the rationales highlights the importance of the final order of the rationales. If the constraint *not-in-defense-zone* is not ordered first, then D_2 may attempt to go to its failing location while still being in the defense zone. This is a behavior we would not want in our soccer robots because there would be a penalty if the robot touches the ball while inside the defense zone. Then, depending on the ordering of *joint-block* and LOC, the robot either moves to the failed location or attempts to block the direct angle of the opposing robot. For this example, we assume the ordering *constraints* → *effects* → *why-this-action*.

Given the ordering, the next step is to loop over the replan policies for the rationale *not-in-defense-zone*. This results in EXIT-DEFENSE-ZONE and CENTRALIZED. Shown in Figure 4.2, D_2 exits the defense zone which makes the constraint, *not-in-defense-zone* valid again. The next step is to loop over the replan policies for the rationale LOC. This results in replan policies MOVE-TO-LOC and CENTRALIZED. The replan policy MOVE-TO-LOC is then executed and D_2 moves to the location in the effect. The rationale LOC becomes valid, while coincidently, the rationale *joint-block* becomes valid again because both D_2 and D_1 are close enough to block the opposing robot using the gap defense. This ultimately end the replanning loop for D_2 (and D_1).

4.3 Summary

In this chapter, we present the intra-robot replanning algorithm. The algorithm uses the rationales provided by the rationale-driven plan to determine when failures or changes in the environment occur that invalidate the current plan of the robot. Using these rationales, the robot is able to determine what replan policies can be used to fix the failure in the plan. With some sorting, a replan policy is chosen and executed to fix the robot's plan. We illustrate this entire process through an example in the robot soccer domain with two defending robots, which highlights the value of the individual robots having access to the rationale for replanning purposes.

So far in this thesis, we have introduced the rationale-driven plan and the intra-robot replanning algorithm. We have provided examples to help illustrate and describe these different parts of the intra-robot replanning problem. Going forward, we explore three domains and provide experimental evidence of the benefits of our approach. Chapter 5 explores the already familiar robot soccer domain. Chapter 6 explores a domain with autonomous underwater vehicles in the ocean. Chapter 7 explores a formally defined planning domain described in the Planning Domain Description Language (PDDL) [Ghallab et al., 1998]. When combined, these chapters highlight the generality of our approach and the unique benefits intra-robot replanning can have in different domains.

Chapter 5

Robot Soccer Domain

In this chapter, we investigate how using our *intra-robot replanning* algorithm, along with some replan policies specific to robot soccer, can enable conditions that are preventing the soccer robot from succeeding at its assigned task. Cooperative multi-robot team planners in competitive robot domains often have limited computational time, have poorly modeled dynamic objects, and have incomplete knowledge of their environment. These issues remain with the individual robots, however we show that the robots can gather state information and improve their reaction to failures by choosing the best replan policy to handle the failure. The centralized team planners could incorporate more information about every robot, but this is not effective or practical for highly dynamic domains with potentially many robots. Likewise, centralized team planners often reduce information and complexity in order to make team planning feasible within dynamic domains, so to much additional information can hinder performance.

The standard approach to team planning in the literature is to use a hierarchy, thereby dividing up the planning problem involved in controlling a team of robots [Yan et al., 2013], [Simmons et al., 2002], [Pecora and Cesta, 2002]. We follow the Skills, Tactics, and Plays (STP) hierarchy as described by [Browning et al., 2005]. This division of computation is used to simplify the problem of controlling a team in dynamic, competitive environments and allows planning to happen at different abstraction layers. Skills are low level repeatable algorithms that are specific to each domain. Tactics combine skills towards accomplishing a more complex task using finite-state machines, consider the Tactic *passing the ball* which includes Skills like driving into the ball, positioning the ball, and kicking the ball. Plays guide the team towards their goal(s). The plays define a Role for each robot that each include a series of Tactics and their input parameters, e.g., positions on the field. Plays are changed based on defined state variables, e.g., the number of robots on the defense side of the field or the ball being on the offensive side. STP essentially separates planning into two levels, global (Plays) and local (Tactics).

In competitive dynamic domains, the global planner assigns a Play using incomplete information and simplified models of the environment. This missing information is often due to opponents' adversarial behavior but can also be attributed to the simplified models of the team members' abilities. A Play only changes when new information is presented at the global level that triggers a failure in its defined state variables, so failures at the local level might not be considered. This is due to the different requirements of the global planner and the local planning robots with respect to their abstraction in planning, i.e., due to local information gains and/or the global planner not fully modeling the domain. We especially see this in competitive domains like the Small-Size Robot

Soccer League (SSL).

The SSL league matches two teams of six omnidirectional robots and each team receives the same information (positions and headings) provided by the Shared Vision System (SSL-Vision) through overhead cameras [Zickler et al., 2009]. Any further information must be generated by each team including velocities, predicting opponents, and the physics of ball movement at the expense of each team's computation. Our centralized planner replans at rate of 60 Hz however the conditions that cause a change at the play level, i.e., the the higher level planner, may change at a slower rate. In STP, a Play may assign a pass between two robots for reasons unknown to the individual robot. Still, the robot is tasked with completing that pass using the assigned Tactic, and opponents may make that exact pass impossible for the assigned robot. Therefore, we provide these assumptions as rationale to the soccer robots, and we show that the soccer robots can monitor these rationale and use the *intra-robot replanning* algorithm to enable failed conditions during execution to make a more successful team.

Note that Plays provide a fully instantiated *team plan* for each robot, T_i, on the team. The robots execute their Tactics according to their assigned variables. The global planner may at any time change the Play due to new information, but there is no guarantee that changes observed only at the Tactic level will cause a change at the Play level. The Play is created assuming optimal conditions for the Tactics and that the robot can successfully execute its Role using the Role's Tactics and their respective parameters. An implicit assumption is that a failure to complete a Tactic leads to a global failure, which the global planner then solves. If however a local failure does not cause an immediate global failure, then the robot continues to fail until complete global failure. In robot soccer, an example of a local failure is a robot maintaining control of the ball but never finding an open pass, which results in it endlessly driving around in circles looking for an open pass. A example of a global failure is a missed pass or stolen ball.

Replanning for individual robots has mostly been seen as a task of minimizing the changes to the current plan, and this same idea has been applied to multi-robot systems. However, in competitive domains, the robots have to compensate for the dynamic nature of the environment, so replanning must be quick and should be focused on enabling failed conditions with the highest chance of success. Optimality is often ill-defined in such complex domains so we focus on the robot accomplishing its assigned Tactic rather than failing, which would subsequently cause a failure of the team's objective.

5.1 Experimental Problem Domain

Passing with marking, shown in Figure 5.1, is a sub-domain of robot soccer where the task is for the robot with the ball, T_b, to pass the ball to the receiving robot, T_r, while opponents try to steal and/or intercept the ball. During our experiments, one opponent is always placed near T_b to block the initial pass. This sub-domain should be familiar as *passing-with-marking* was described in Chapter 2. However, in this chapter, we have two opponents, O_b and O_r, that are placed on the line between the two T_i to block and intercept the direct pass. Specifically, the opponent O_b attempts to steal the ball while the other opponent O_r attempts to intercept passes.

An example Play generated by the global planner is given in Figure 5.2. The Play is made of Tactics: *Pass*, *Goto*, and *Receive*; the *Initial Position* defines the robots' starting positions. Each Tactic has instantiated variables defining the assigned robot and the variable(s) for that Tactic. For

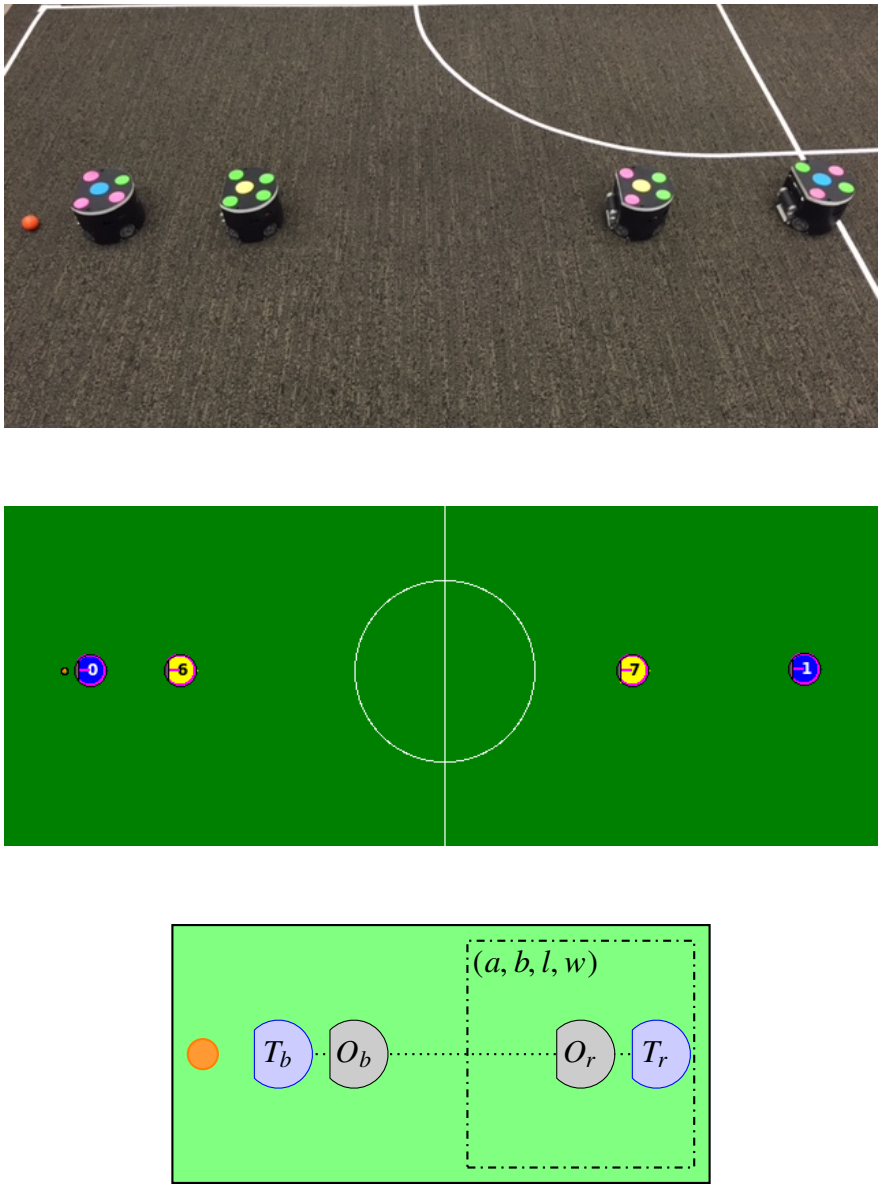

Figure 5.1: Passing with marking: T_b needs to pass the ball to T_r within the zone (a, b, l, w). T_b is facing the ball. Opponents O_b and O_r try to remain on the line between the other robots to block or intercept the pass. The top image is the physical robots, the middle image is the robots in the physics-based simulator, and the bottom image is a simplified representation that we use in this chapter.

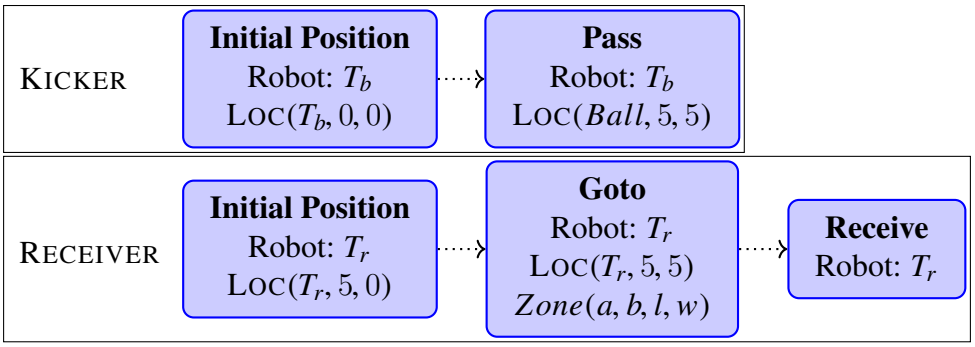

Figure 5.2: A team plan for T_b to pass the ball to T_r.

Pass, T_b is assigned to kick the ball to $(5, 5)$. For *Goto*, T_r is going to location $(5, 5)$ to *Receive* the ball and must stay within the $Zone(a, b, l, w)$ defining a box with the left top corner at location (a, b) with length l and width w.

The Play assigned the Role KICKER to T_b and RECEIVER to T_r. In this case, T_r is assigned the best passing location within its zone (the dashed-dotted square in Figure 5.1), but clearly O_r will attempt to intercept the ball. The assignment for the best passing location and the generation of the probability of success for that pass follows the approach in [Biswas et al., 2014]. The location and probability are based on the probability that the receiver will successfully receive the ball at that location and the probability that the receiver will successfully score a goal from that location. Obtaining those exact probabilities in the highly dynamic and complex domain of robot soccer is unrealistic, and therefore the authors make some simplifications and assumptions. For our purposes, the rationale for the chosen location is the combined estimated probabilities, and a more thorough understanding of the estimated probability can be found in [Biswas et al., 2014]. The Play often fails from the opponents' interferences. It also fails due to randomness in the robot's performance and kicking ability in the physics-based simulator. There have been attempts to solve this issue in robot soccer as passing is a major requirement, and machine learning has been a major focus of this research [Stone et al., 2006]. However, our approach is to use intra-robot replanning with designed replanning algorithms (replan policies) that enable failed rationales provided by the centralized team planner.

5.2 Individuals Enabling Conditions

In [Mendoza et al., 2016], they describe an approach called pass-ahead that tries to sync the arrival of the receiver with the ball so that they arrive at the passing location at the same time. This approach alone does poorly in the *passing with marking* domain because of the opponent O_b that blocks or steals the ball. In Chapter 2, we demonstrate the requirement of opponent aware algorithms for T_b to maintain possession of the ball, which improves pass-ahead's performance in the *passing with marking* domain (**Dribbling-Move** in Results Section 5.3) [Cooksey et al., 2016]. However, that algorithm does poorly in our example domain given that an opponent is added to mark the receiving robot. The performance is poor because Dribbling-Move only focuses on replanning to enable one *constraint* of the Tactic **Pass**, which is ball possession ($HasBall$). $HasBall$ is defined as true for a robot if the ball is closer to it than any other robot. And, the condition being

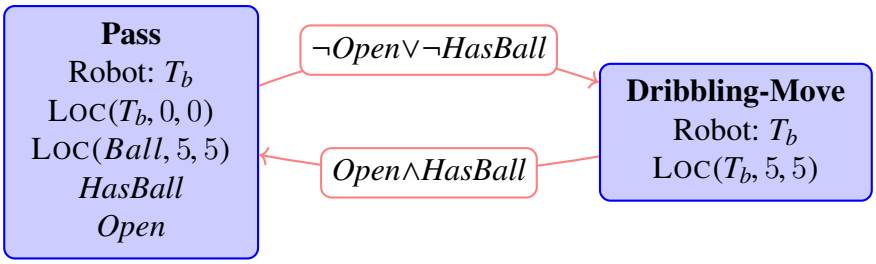

Figure 5.3: *Intra-robot replanning* to enable Pass's rationale through dribbling. *HasBall* and *Open* are conditional functions (T/F) and when either is invalid the replan policy **Dribbling-Move** is executed to enable them.

failed is $Open$, which is defined as true if the pass can be successful. $Open$ is ill-defined without knowledge of the opponents' intercept abilities and therefore is estimated by the centralized planner. Another issue is in T_b's ability to enable the $Open$ condition.

Dribbling-Move enables the condition $HasBall$ by dribbling – the robot has a rotating bar in its forward direction that applies a back spin on the ball for maintaining possession – and can implicitly enable the condition $Open$ by moving the ball towards the passing location while circumventing nearby opponents. This does not solve the issue of opponents marking the receiver or the ill-defined $Open$ condition. For now, $Open$ is defined as true if the probability of a successful pass, given by the centralized team planner, is above the defined threshold γ. The flow diagram in Figure 5.3 illustrates the *intra-robot replanning* for T_b using Dribbling-Move. The robot changes its local position variable towards the passing location. As previously stated, the condition $Open$ is only being enabled implicitly as T_b is not attempting to find a new or better passing location. The condition is enabled when the centralized planner declares the current pass probability above γ given some assumptions and simplification based on the current state. Dribbling-Move gives a relatively simple solution to enabling conditions $HasBall$ and $Open$, however we show soon that it performs poorly in our example domain.

$HasBall$ can be defined as an **independent condition**. The ability to enable it is on a single robot's own ability to keep the ball or get the ball. Independent conditions can be enabled by changes in the robot's assigned variables to compensate for changes in the environment. $Open$ can be defined as a **dependent condition**. Its ability to be enabled is highly dependent on the other teammate(s) and/or opponent(s). Dependent conditions can be enabled by changes in the variables of the robots that are involved with enabling the condition. Obviously, the opponents' variables cannot be manipulated directly so it can be difficult to enable a dependent condition.

To enable $Open$, T_b needs to change its variables – this occurs through dribbling – and the variable(s) of the teammate's Tactic involved in the pass. Here, we add the action **Adjust Teammate** to Dribbling-Move which changes the $Zone$ variable of the receiver's Goto Tactic, shown in Figure 5.4. This replan policy communicates the change of the zone variable with the other robot while the other replan policy, Dribbling-Move, only dribbles the ball. Adjust Teammate is used if $Open$ is not enabled when dribbling. It places the zone in front of T_b and towards T_r as shown in Figure 5.5. A new location is found within the new zone for the pass and provided to T_b and T_r. T_b can now explicitly attempt to enable the $Open$ condition by moving the teammate to a zone better situated for its own dribbling abilities. This ultimately helps the robots work together towards their shared goal of passing the ball.

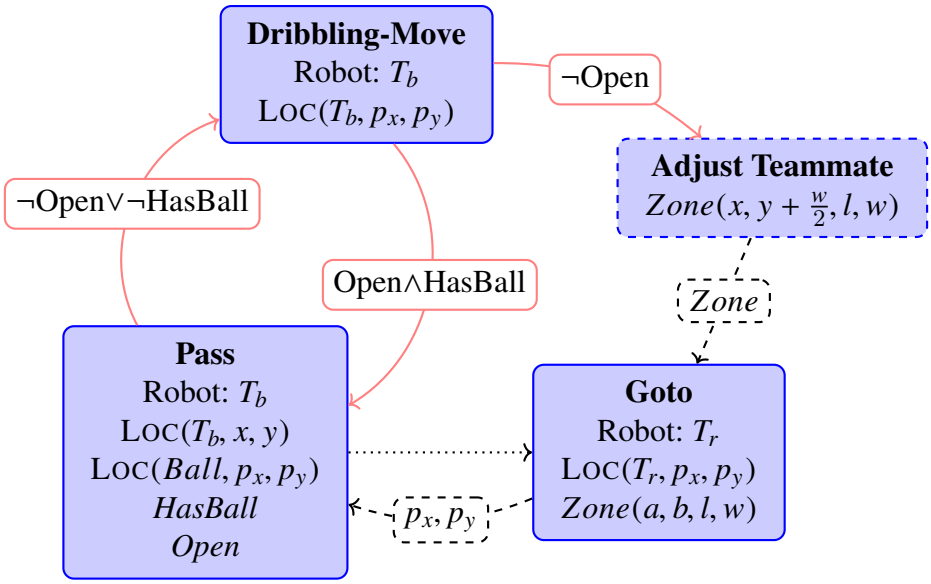

Figure 5.4: *Intra-robot replanning* to more explicitly enable *Open* by using the action Adjust Teammate to change the zone variable.

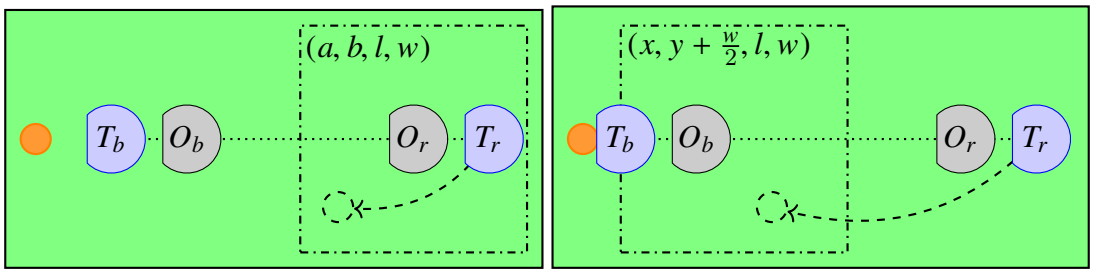

Figure 5.5: Demonstrating the *Zone* adjustment (dot-dash black square), resulting in a change in pass location (dashed black circle), from the replan policy in Figure 5.4.

The *Zone* becomes a sliding window towards the right based on T_b's location, shown in Figure 5.5. This ensures that T_b is always close to the passing *Zone* and the centralized planner finds T_r a new pass location.

5.3 Experimental Results

5.3.1 Experimental Setup

We use the *passing with marking* domain as we describe in our example with a field size of $9\ m$ long by $6\ m$ wide. T_b starts at $(2\ m, 3\ m)$ with the ball directly in front of it at $(1.9\ m, 3\ m)$ and opponent O_b behind it at $(2.4\ m, 3\ m)$. Teammate T_r starts at $(7\ m, 3\ m)$ with the opponent O_r in front of it at $(6\ m, 3\ m)$. The Play is always the same with T_b passing to T_r, however the passing location changes quickly as the centralized planner attempts to find the best position within T_r's defined zone. The zone is defined as $(x = 4.5\ m, y = 0\ m, l = 4.5\ m, w = -6\ m)$, so the zone covers

half of the field.

The pass must happen within one minute or it is considered a failed pass. If the ball goes outside field boundaries it is also considered a failed pass. Touching is defined as being within a robot radius plus a ball radius of the center of a robot. If the ball is touching an opponent for five video frames then it has been intercepted and the pass has failed. A pass succeeds if T_r is touching the ball for five video frames.

The opponents place themselves in the direct line of sight between the two T_i. O_b maintains a close distance of three robot radii $(3 * 0.09\ m)$ from T_b. O_r maintains a distance of $0.5\ m$ from T_r. When the ball is kicked, i.e., its velocity goes above $0.9\ m/s$ and it is two robot diameters away from T_b, O_r attempts to intercept the ball.

T_r uses the pass ahead algorithm to receive the ball as previously referenced. We are introducing **Change-Zone** which follows Figure 5.4. The zone parameters are determined by trial and error for improving the performance of passing. We compare **Change-Zone** to three algorithms for T_b with varying degrees of replanning to enable invalid rationale:

- **Base Case**: Positions to face the ball and pass location, then immediately kicks as planned (no replanning).
- **Dribbling-Move, D_M^0, $\gamma = 0$**: Previously referenced algorithm with γ set for any pass.
- **Dribbling-Move, D_M^1, $\gamma = 0.15$**: Only passes if the pass probability is above γ.
- **Change-Zone, C_Z**: Uses D_M^1 with the Adjust Teammate action to enable the condition $Open$ by altering the Teammate's Zone using: $(l = 2\ m, w = 6\ m)$.

5.3.2 Results

Each different replan policy for T_b is run five hundred times in a physics-based simulator starting with the same formation of teammates, opponents, and the ball. The results are found in Figure 5.6. Base Case fails almost every time and its few successes can be attributed to random chance as the ball is often stolen or blocked. Both Dribbling-Move algorithms improve the chance of a successful pass by enabling the constraint $HasBall$, but passes are often intercepted by O_r. With the higher γ, Dribbling-Move further increases the success rate by waiting for γ to improve (even if just slightly as 0.15 is still very low). Change-Zone further increases the success rate by more explicitly attempting to enable $Open$ by moving T_r closer to find an open pass.

We use the binomial proportion test to verify that the different success rates of the algorithms are not due to randomness or limited sampling. The binomial proportion test is, as defined in [Ryan, 2008]: given a set of N_1 observations in a variable X_1 and a set of N_2 observations in a variable X_2, we can compute a normal approximation test that the two proportions are equal, or alternatively, that the difference of the two proportions is equal to 0. A standard approach is that at a p value of less than 0.05, the null hypothesis can be rejected. In our case, the null hypothesis defines that there is not enough evidence to show a statistically significant difference between the results of the algorithms.

As shown in Figure 5.7, each improvement on the Base algorithm is statistically significant when compared to previous algorithms. The initial change, D_M^0, occurs when replanning enables the condition $HasBall$ by dribbling. Our contribution, C_Z, shows further significant improvement

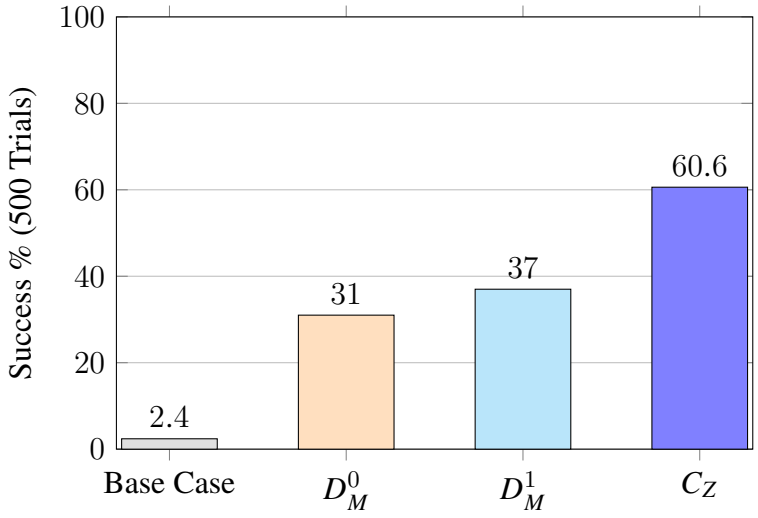

Figure 5.6: Results for four different replan policies used by T_b in the *passing with marking* domain shown in Figure 5.1.

Binomial Proportion Test (Two-Tailed)		
Algorithm	**Algorithm**	**P Value**
Base Case	D_M^0	< 0.0001
D_M^0	D_M^1	0.0452
D_M^1	C_Z	< 0.0001

Figure 5.7: Testing for the null hypothesis that the two proportions are equal and the difference is due to randomness.

by replanning to enable the condition $Open$ by moving the teammate into a more beneficial zone. Our data demonstrates the large improvement when individual robots within a team can replan to enable all the conditions needed by their Tactics.

In Chapter 2, we show that the Dribbling-Move algorithm improves the success rate of passing when marked by an opponent similar to O_b. With the inclusion of the second opponent, O_r, the success rate drops dramatically. We can see that the improvement of Dribbling-Move mainly contributes to the enabling of the condition $HasBall$ and its ability to handle the close opponent, O_b. Changing-Zone's improvement is in changing the teammate's zone variable to support the robot's dribbling abilities. Changing the zone variable brings the passing location closer for tighter, shorter passes and this helps improve the probability of enabling $Open$. Overall, this improvement occurs because we provide the individual robot with a new replan policy that attempts to enable a rationale that previous replan policies had not explicitly handled.

In Figure 5.8, we change the initial locations of the soccer robots. In this experiment, the success rate of Dribbling-Move and Change-Zone are comparable, and the base case method performs much better. This experiment is meant to demonstrate that the replan policies can be better or worse in different states of the world. This further motivates our work in Chapter 8 in which we learn a value metric for each replan policy in order to sort them and further improve team performance.

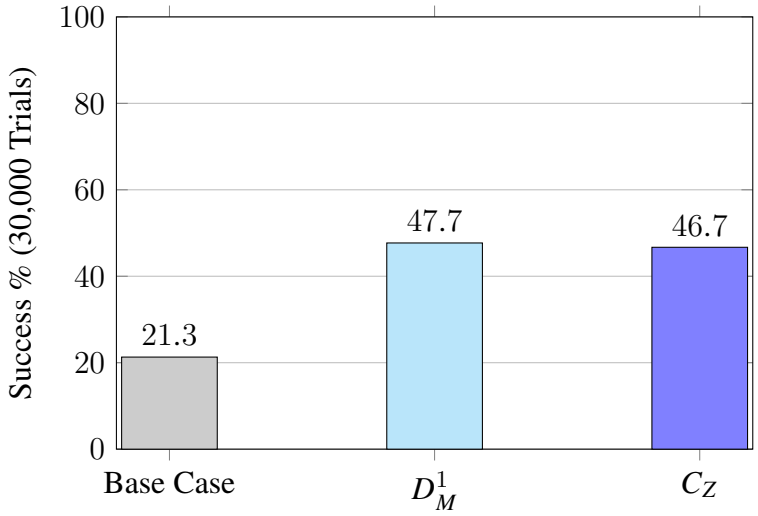

Figure 5.8: Results for three different replan policies used by T_b in the *passing with marking* when starting locations are randomly chosen in Figure 5.1.

5.4 Summary

In this chapter, we present experimental results in the *passing-with-marking* domain, a sub-domain of the robot soccer domain, using intra-robot replanning. We discuss the inability for the previous methods to successfully pass the ball because of the introduction of another opponent, which made this sub-domain more closely resemble a realistic passing scenario in a robot soccer game. With the introduction of a new opponent, the inability of previous methods to enable the *open* constraint is clear. Therefore, we introduce a new replan policy that alters the zone variable of the receiving teammate in order to make passing more successful. The results show a clear improvement over the previous methods and highlight the importance of having replan policies that handle all the rationales provided by the centralized planner.

In the next chapter, we explore a domain vastly different than the robot soccer domain; a domain where the vast, deep ocean is the environment, where the robots are autonomous underwater vehicles (AUVs), and where communication between robots and the centralized planner happens by a satellite link. Although the domain is different, our intra-robot replanning algorithm with our rationale-driven plan improves the AUVs' abilities to handle failures locally, and ultimately they improve the team's performance.

Chapter 6

Autonomous Underwater Vehicle Domain

In this chapter, we consider multiple autonomous underwater vehicles (AUVs), shown in Figure 6.1, engaged in oceanic missions that require collaboration. We address the challenge of effective distributed multi-robot plan execution: when a failure occurs, an individual AUV needs to decide how to proceed to maintain the success of the multi-robot plan. Failures occur when the modeled actions of the AUVs fail to result in their expected outcomes, largely due in this domain to the poorly modeled dynamics of the ocean. Intra-robot replanning is advantageous because traditional purely centralized controlled methods use time-consuming and expensive satellite communication in order to repair failures. We use our rationale-driven team plan representation that explicitly includes the rationale of the individual robots' planned actions to enable local replanning without satellite communications. This chapter focuses on examples of concrete team missions and the collaboration of AUVs that clearly demonstrate the effectiveness of our approach.

In the AUV domain, we assume that a centralized controller, located offshore, generates team plans for a team of AUVs. Human operators create the team plans in this domain and add the necessary rationales. We assume the AUVs are placed into the ocean having already received the team plan, and from then on the only communication between the AUVs and the centralized controller is through a satellite link which is expensive and time consuming to use. The AUVs can quickly and cheaply communicate with each other over wifi if they are close enough in proximity

Figure 6.1: Three Light Autonomous Underwater Vehicles (LAUVs) from the Laboratório de Sistemas e Tecnologia Subaquática (LSTS) in Porto, Portugal that were used to gather the depth data in Figure 6.4.

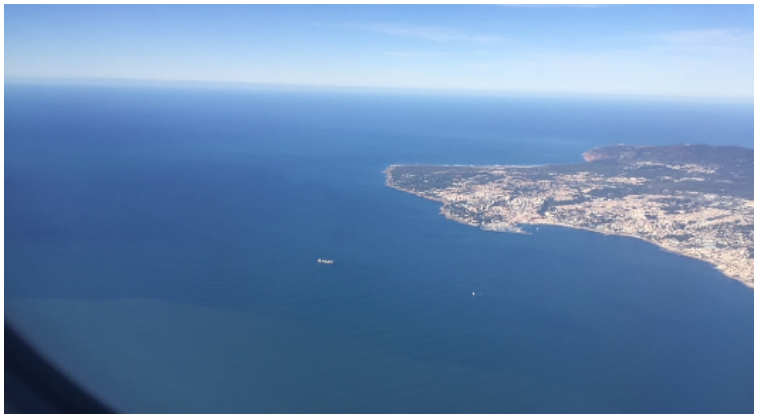

Figure 6.2: Coastal waters off of southern Portugal. Discoloration in the ocean, near bottom left corner, shows the outline of an ocean front between river runoff and ocean water.

and are not underwater. To reiterate, the AUVs cannot communicate underwater or receive a GPS signal for localization while underwater. The ocean also has many underwater currents that are very hard to model and predict. Therefore, the AUVs often fail to arrive at their destination when traveling underwater because they cannot correct for the unknown current without a GPS signal. Ultimately, this causes the team plan to fail. Further discussion of the AUV domain is provided in Section 6.1.

In order for autonomous intra-robot replanning to be successful within a team, the individual AUV must understand the rationales of the centralized planner for the actions being executed. Furthermore, once the rationales are known, the individual AUV must know which replan policies can enable the failing rationales of the team plan. Finally, the individual robots must select a single replan policy from those applicable ones with two considerations: most importantly, enable the failing rationale given the state of the environment, and, secondly, do so in a cost-effective manner.

6.1 Domain

In this chapter, we are using a simulated AUV domain where two AUVs follow a defined path together (simultaneously), criss-crossing the boundary line between fresh and salt water in order to collect multiple data points in the water column for oceanography research. An example of such a boundary line in the ocean is shown in Figure 6.2. The simulated path we use in our domain is shown in Figure 6.3. The AUVs complete a yoyo maneuver by ascending and descending multiple times between two points on the ocean surface. In order to collect more sensor data in the water column, the AUVs time their movements to have inverse motion in the z-axis (depth). Shown in Figure 6.4 is depth data collected by two real AUVs completing the described yoyo maneuver between two points on the ocean surface. This data was collected using two of the AUVs shown in Figure 6.1 through a collaboration with Laboratório de Sistemas e Tecnologia Subaquática (LSTS) [Ferreira et al., 2018].

Our concerns with this domain are:

1. The AUVs must connect at each location in order to synchronize their movements so that they can perform the behavior shown in Figure 6.4;

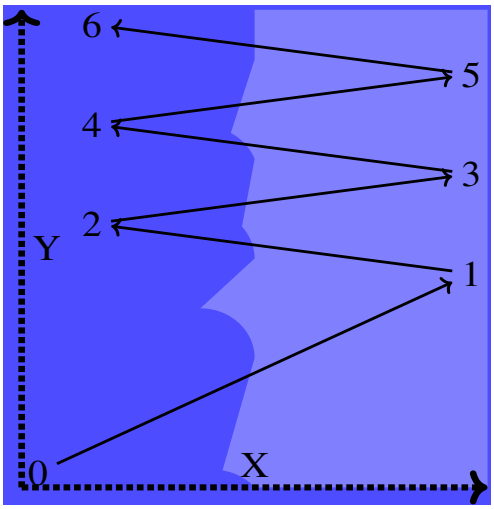

Figure 6.3: A simulated environment where two AUVs criss-cross the boundary line of fresh water (light blue) and salt water (dark blue) to gather scientific data about the boundary. At each location the AUVs connect to each other and synchronize their movements and then they yoyo to the next location. See Figure 6.4 for the yoyo maneuver between two points.

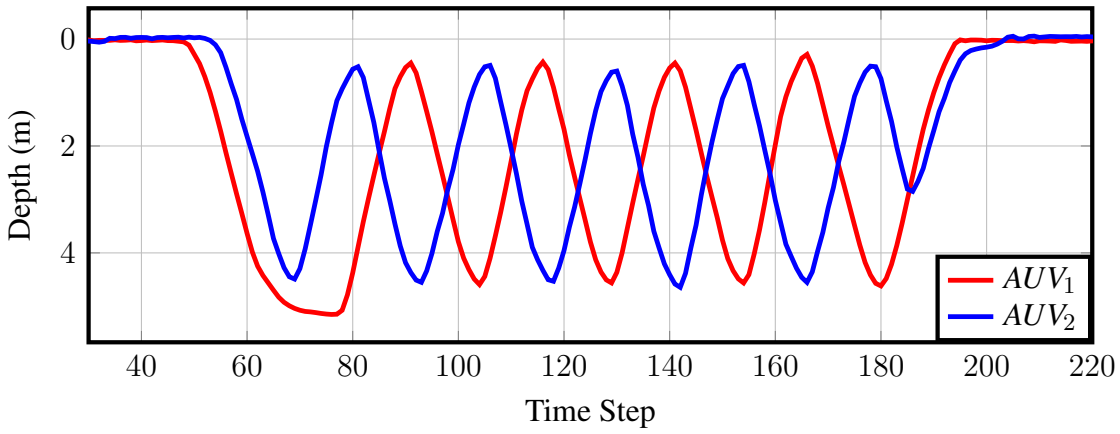

Figure 6.4: Real depth data collected by two AUVs. They sync their movements before time step 50 then execute a yoyo maneuver where one AUV is at the top of the water column while the other is at the bottom of the water column. After a set number of yoyos, they ascend at time step 200. They must connect again after step 200 before moving to the next location.

2. Unknown water currents often push the AUVs off their intended course while they perform the yoyo maneuver;

3. Communication for replanning between the AUVs and the centralized controller by satellite communication is very costly.

To handle these concerns, we implement intra-robot replanning with our rationale-driven plan representation to handle failures locally whenever possible.

6.2 Simulated Experiments

We develop a simulated environment in order to test the rationale-driven team plan representation with the intra-robot replanning algorithm over multiple experiments with different environmental conditions. In this section, we describe:

1. The constants of the experiments,

2. The actions used by the AUVs,

3. Examples of rationales used in the rationale-driven plan,

4. The replan policies used by the AUVs,

5. Computation and communication costs

6. The experiments and their results.

6.2.1 Constants

Everything in this simulation is defined in kilometers and seconds. The constants below are assumed:

- AUV speed is 0.0128 km/s

- Wifi radius is 0.1 km

- Satellite cost is 600 seconds (10 mins)

- Wifi cost is 3 seconds

- Maximum wait time ($\mathbb{W}$) for a wifi connection is 300 seconds (5 mins)

6.2.2 Actions

The actions used in our experiments are:

- INITIAL: Starting action for the AUV.
- SYNC ⟨ID⟩: Connects over wifi to the AUV with the defined ID and synchronizes their next action.
- GOTO ⟨X⟩ ⟨Y⟩: The AUV travels along the surface and heads directly to the (X, Y) location. We assume it can receive a GPS signal so it does not get lost. However, the AUV cannot receive a wifi signal unless it stops because the AUV's own motion and the surrounding waves make it harder for the wifi signal to propagate horizontally.
- YOYO ⟨X⟩ ⟨Y⟩: The AUV performs a series of descents and ascents in the water column towards the (X, Y) location. As the AUV is underwater, it cannot receive a GPS or wifi signal so the YOYO maneuver often fails to arrive at the destination location.
- SATELLITE: The AUV connects to the centralized controller through a satellite link.

6.2.3 Rationale

The rationales used in our experiments are provided by the human operator developing the team plan. We describe some example rationales for the AUV domain. For our experiments, we use only the last rationale described in this section, i.e., CANCONNECT.

Consider the AUV must take a water sample precisely $AT\{X = 1, Y = 2\}$, a typical constraint in planning. We can define this constraint as the following rationale:

$$\text{AT}(X, Y, 1, 2) = X \equiv 1 \wedge Y \equiv 2 \tag{6.1}$$

where X and Y are the location variables of the AUV. This rationale is only true if the AUV is at that specific location. The two constants (1 & 2) can be replaced with any location if the rationale for the plan defines that the AUV must be at a defined location.

Consider the human operator wants to be more lenient with the location and allows the AUVs to be near enough to the location. We could define the NEAR rationale:

$$\begin{gathered}\text{NEAR}(x, y, l_x, l_y, d_x, d_y) = \\ (|x - l_x| < d_x) \wedge (|y - l_y| < d_y)\end{gathered} \tag{6.2}$$

where x, y are the AUV's location, l_x, l_y is the destination location, and d_x, d_y are the distance thresholds.

Consider two or more AUVs must connect over wifi to synchronize before moving to the next location. We assume the AUV's state ($\mathcal{S}$) defines if the AUV is connected over wifi ($\mathbb{C}$) and to a defined ID ($\mathbb{C}_{ID}$). We define the CANCONNECT rationale:

$$\begin{gathered}\text{CANCONNECT}(\mathbb{C}, \mathbb{C}_{ID}, ID) = \\ (\mathbb{C} = true) \wedge (\mathbb{C}_{ID} = ID)\end{gathered} \tag{6.3}$$

These rationales add information about why certain locations were picked (to fulfill AT or NEAR), or about an assumption made by the human operator regarding the ability of both AUVs to connect with one another.

6.2.4 Replan Policies

When designing the replan policies for the AUVs, the rationales need to be considered. There is an assumption that there exists at least one replan policy that enables each rationale. The basic replan policy simply calls the satellite, but we also considered another alternative:

R-SATELLITE ⟨β = any rationale⟩: Uses SATELLITE to receive a new plan from the current robot's location (A).

R-GOTO-SATELLITE ⟨β = any rationale⟩: Uses GOTO to move the AUV to the correct location (B) and then uses SATELLITE.

In this chapter, we only consider the CANCONNECT rationale for the experiments. We provide a few different options that attempt to reconnect with the other AUV when it fails to connect over wifi:

R-GOTO ⟨β = CANCONNECT()⟩: Uses GOTO to move to the correct location (B) and then uses SYNC to attempt to connect with its teammate over wifi.

R-CONNECT-GOTO ⟨β = CANCONNECT()⟩: Attempts to use SYNC first, and then proceeds like R-GOTO.

R-STAR-GOTO ⟨β = CANCONNECT()⟩: Attempts SYNC. Then centered at its current location, it creates five evenly spaced points on the circumference of a circle with a radius equal to the wifi distance. It will GOTO each location and attempt to SYNC. The first location is chosen at random and then it always proceeds to the point farthest away from its current location until all points have been attempted. If unsuccessful, it proceeds like R-GOTO. This is inspired by the sector search pattern described in [Bernardini et al., 2017] for search and rescue, but is modified to ensure that the AUVs always travel the same distance before attempting to connect.

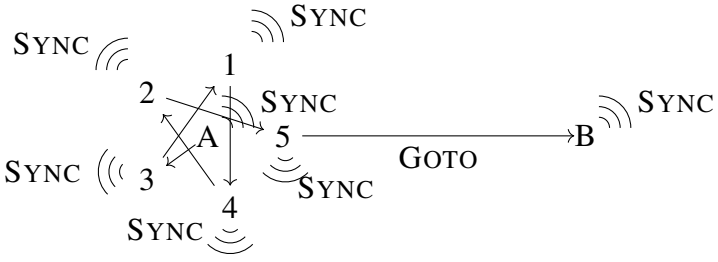

Table 6.1: Team Plan Provided to AUVs with Rationales

AUV1, ID = 1	**AUV2, ID = 2**
INITIAL	INITIAL
SYNC 2	SYNC 1
YOYO 8 4; CANCONNECT(2)	YOYO 8 4; CANCONNECT(1)
SYNC 2	SYNC 1
YOYO 1 5; CANCONNECT(2)	YOYO 1 5; CANCONNECT(1)
SYNC 2	SYNC 1
YOYO 8 6; CANCONNECT(2)	YOYO 8 6; CANCONNECT(1)
SYNC 2	SYNC 1
YOYO 1 7; CANCONNECT(2)	YOYO 1 7; CANCONNECT(1)
SYNC 2	SYNC 1
YOYO 8 8; CANCONNECT(2)	YOYO 8 8; CANCONNECT(1)
SYNC 2	SYNC 1
YOYO 1 9; CANCONNECT(2)	YOYO 1 9; CANCONNECT(1)
END	END

For the previous three replan policies, an attempted SYNC lasts 3 seconds and it uses the ID provided by the rationale.

6.2.5 Computation and communication costs

The intra-replanning algorithm is computationally efficient for an AUV. The replan policies are simple state-machines that are selected by their applicability and their cost. This should not be very taxing for the on-board computation of the AUVs. The communication cost is slightly increased when the team plan is delivered to the AUVs due to the addition of rationales which come with each action. Although, the communication cost saved from replanning is significant enough to outweigh the addition of some communication overhead due to the addition of team plan rationales.

6.2.6 Experiments

For our experiments, we follow our example domain with two AUVs criss-crossing the ocean using the team plan shown in Table 6.1 and assume the AUVs start at location (0, 0). A visualization of the plan's path is in Figure 6.3. We assume that there exists a current that pushes the AUVs off course when they are performing the YOYO maneuver. The simulation uses a simplified model of currents that are only applied to the AUVs' locations after the simulation has determined the YOYO maneuver finished. Consequently, the AUVs surface near the intended destination based on the currents' modeled distributions. There has been work on modeling ocean and wave currents for real-time simulations, but this level of simulation goes beyond the scope and need of this work [Omerdic and Toal, 2007].

In this domain, each AUV completes a set number of yoyos and then surfaces to connect with the other AUV before beginning the next traversal. The only purpose of the specific locations is to direct the AUVs in a certain direction and so the only rationale added to the team plan is that they

need to be able to connect when they surface. The defined location does serve as a location known by both AUVs in case they cannot connect where they surface, which is ultimately how our replan policies are designed to connect the AUVs if all else fails.

We therefore add CANCONNECT to every YOYO tactic in the team plan shown in Table 6.1. See below for a general example:

YOYO ⟨X⟩ ⟨Y⟩; CANCONNECT(⟨ID⟩);

Experiment 1: Positive Y-Axis and X-Axis Currents

We assume there is an ocean current in the positive y-axis modeled by a normal distribution with mean 2 and variance 0.1, and a uniform distribution from 0 to 0.1 in the x-axis. See Figure 6.3 for the orientation of the axes. Figure 6.5 shows the time taken by the AUVs to complete the team plan using a specific replan policy for every CANCONNECT rationale that became invalid. The first three replan policies have very consistent performances because they always go towards the failed location and are affected only slightly by the ocean current. Relying on satellite communications for every failure is clearly very costly. The intra-robot replanning policies that attempt to connect over wifi perform much better. However, the AUVs have a limited wifi radius of 0.1 km and so they cannot always connect when they surface. R-STAR-GOTO attempts to find the other AUV but the search patterns can add more time, which is why its performance has a higher variance. On the other hand, R-CONNECT-GOTO only adds wifi costs to its time in comparison with R-GOTO.

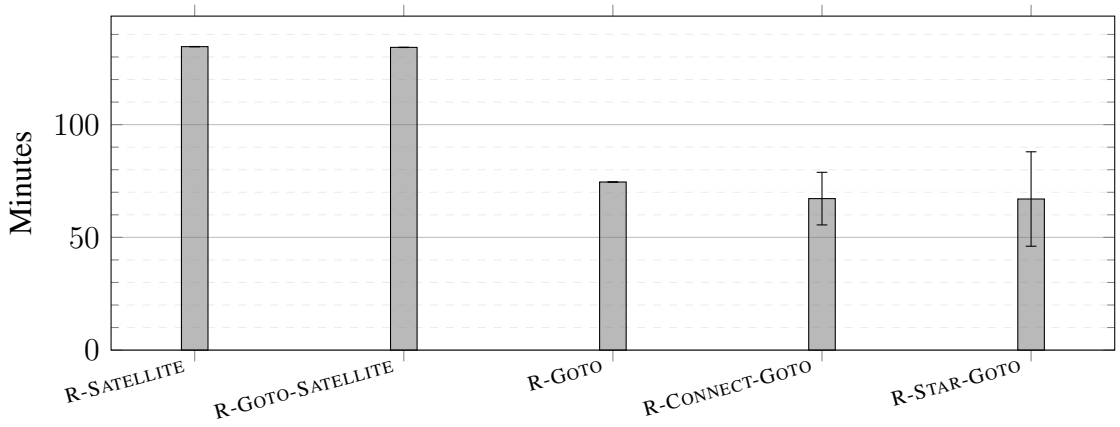

Figure 6.5: *Experiment 1, ocean current in both x- and y-axes directions*: Average time and variance to complete the team plan 5,000 times using the stated replan policy.

Experiment 2: Only Positive X-Axis Current

We assume there is an ocean current in the positive x-axis modeled by a normal distribution with mean 1 and variance 0.05, and no current (0) in the y-axis. Figure 6.6 shows that R-CONNECT-GOTO and R-STAR-GOTO perform reliably better than R-GOTO because the two AUVs are often within the wifi radius after surfacing from a yoyo. Similar to the previous experiment, R-STAR-GOTO has a higher variance because of the extra search time. This experiment has a less aggressive current and the variance is less, so the overall performances are much more consistent.

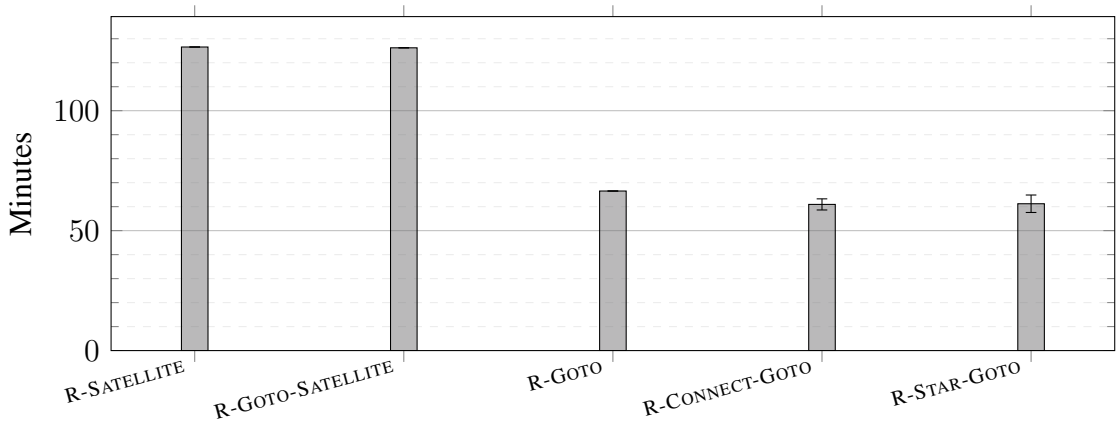

Figure 6.6: *Experiment 2, ocean current in the x-axis direction*: Average time and variance to complete the team plan 5,000 times using the stated replan policy.

Experiment 3: Wifi Not Available

We assume there is an ocean current in the positive x-axis and y-axis both modeled by a normal distribution with mean 1 and variance 0.2. We are assuming the surface waves are so large that connection over wifi **cannot** be established. The replan policies that try to connect always fail at the maximum wait time on their final SYNC and then R-SATELLITE is executed. Therefore, their average times are consistent, and the time variance is mainly due to the increase in the variance of the ocean current. Shown in Figure 6.7, R-GOTO-SATELLITE is the best option because it first moves to the correct location and then immediately calls the satellite. Because both AUVs are together, the centralized controller connects them and they start their next yoyo. R-SATELLITE, however, connects to the satellite at the failed location, receives a plan to go to the failed location (to bring the AUVs together), and then cannot connect over wifi at the failed location. Therefore, it must call the satellite again, causing it to be the worst option by far.

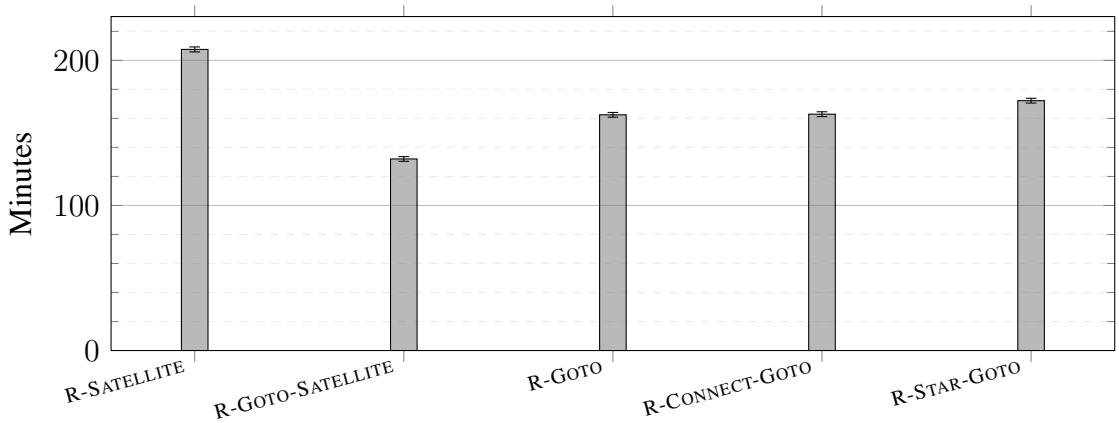

Figure 6.7: *Experiment 3, large ocean currents with no wifi connections*: Average time and variance to complete the team plan 5,000 times using the stated replan policy.

Review

These experiments show that a predetermined sorting of the intra-robot replanning policies in one environment is not always the best option in another environment. Another challenge is handling unknown variables when deciding the order of the replan policies. In the Chapter 8, we describe a method to learn the predicted cost of each replan policy in order to more reliably select the lowest costing one.

6.3 Understanding the Effects of the Constants

In this section, we explore the effects of the constants on the performance of the replan policies and the AUVs' plan completion time. First, we reiterate the assumed constants for the AUV domain – please note that the AUV's speed has been lowered for these experiments. The experiments are as follows: (i) explore the effect of the satellite communication costs, (ii) explore the effect of the modeled ocean variance, (iii) explore the effect of the maximum wait time, and (iv) explore the effect of the wifi range.

6.3.1 Constants

Everything in the simulation is defined in kilometers and seconds. The constants below are assumed unless otherwise discussed in each individual experiment:

- AUV speed is 0.00128 km/s
- Wifi radius ($\mathbb{R}$) is 0.1 km
- Satellite cost ($\mathbb{S}$) is 600 seconds (10 mins)
- Wifi cost is 3 seconds
- Maximum wait time ($\mathbb{W}$) for a wifi connection is 300 seconds (5 mins)

6.3.2 Satellite communication costs

We run multiple experiments to test the effects of different satellite costs to show the increasing benefits of intra-robot replanning. For each experiment, a single replan policy is run two hundred times and then the team plan completion times are averaged. We assume the satellite replan policy is still available if the other replan policies fail. The satellite cost is then changed and the process repeated for each replan policy. The x-axis ocean current is modeled with a normal distribution with mean 1 and variance 0.02, and the y-axis has no ocean current. The large x-axis current pushes the AUVs off their intended target locations, forcing the AUVs to replan.

As shown in Figure 6.8, the cost of communicating with the satellite can greatly increase the time for the AUVs to complete their assigned team plan. On the other hand, the costs of replan policies R-GOTO, R-CONNECT-GOTO, & R-STAR-GOTO remain constant because they can successfully replan without needing to connect to the satellite. Although, their costs could be affected by the types of currents and their variances in our modeled ocean currents.

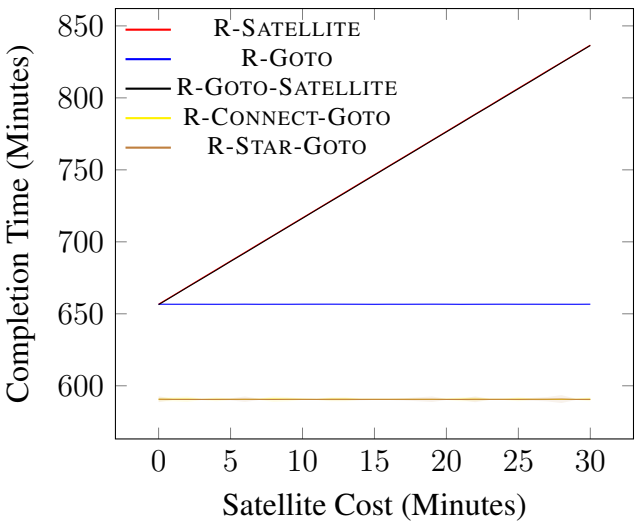

Figure 6.8: The relationship between team plan completion time and the satellite cost used when replanning for each replan policy. Brown and yellow lines are nearly identical, as are black and red. See Section 6.3.2.

6.3.3 The effect of modeled ocean variance

In the previous experiment, we demonstrate that some replan policies can greatly reduce the time it takes to complete a team plan, but in those experiments the ocean current's variance is static at 0.2 km. In this experiment, we vary the ocean current variance to see the effect on the overall performance of each replan policy. We model both the x-axis and y-axis currents by a normal distribution. Both have a mean of 0 and the variance goes from 0 to 0.5 km stepping by 0.02 km. The experiment is run 400 times for each replan policy for each variance.

Shown in Figure 6.9, the increase in ocean variance steadily elongates the team plan completion time. Of course, with more ocean variance the AUVs may have to travel farther, automatically increasing the total time. The graph shows that as the ocean variance increases, the AUVs are less likely to be successful when using the replan policy. The shaded area around the graph lines show one standard deviation around the average, which increases as the ocean variance increases. Surprisingly, the performance of R-STAR-GOTO degrades quickly, eventually crossing beyond R-SATELLITE. This demonstrates that R-STAR-GOTO is not very useful with large variance in the ocean currents. R-CONNECT-GOTO and R-GOTO never reach the cost of R-GOTO-SATELLITE so we can assume they successfully replan sometimes even with the very large ocean variance. An interesting aspect of the graph is the u-shaped curve between 0 and 0.2 variances for R-SATELLITE. The initial sharp increase highlights the quick increase in cost when the ocean variances gets just large enough to cause failures for the AUVs. However, the cost goes down afterwards for awhile and then picks back up. We explore why in the next section.

6.3.4 Maximum wait time ($\mathbb{W}$)

We notice a rather interesting phenomenon about the simple constant $\mathbb{W}$. Maximum wait time ($\mathbb{W}$) is the time that the AUVs attempt to connect over wifi before failing and calling the satellite. In the

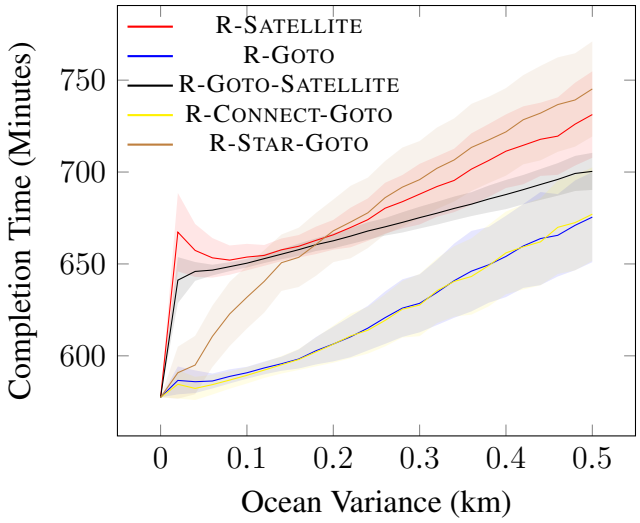

Figure 6.9: The relationship between team plan completion time and variance of the ocean currents for each replan policy. Highlighted areas show 1σ standard deviation. See Section 6.3.3.

previous experiment, we describe a u-shaped curve in the performance of R-SATELLITE that we attribute to $\mathbb{W}$ in this section. For this experiment, we simplify the ocean current to only a normal distribution on the x-axis with mean 0 and variance ranging from 0 to 0.25 km incrementing by 0.02 km. $\mathbb{W}$ varies from 30 seconds to 5 minutes incrementing by 30 seconds. The experiment is run 400 times for each ocean variance and $\mathbb{W}$.

In Figure 6.10, we show the results of R-SATELLITE where each line has a different $\mathbb{W}$ value. Reducing the ocean variance to one axis reduced the u-shape curve but it remains prominent. Clearly, $\mathbb{W}$ causes the u-shape as it goes away once the value is very low. There are three cases to consider: (i) both AUVs are near the target location, (ii) only one AUV is near the target location, or (iii) both AUVs are far from the target location. Consider the cost of the two AUVs reuniting using R-SATELLITE. The first case has cost zero because they are both correctly at the target location, i.e., no reunion cost. The next two scenarios are more complicated and their combination results in the u-shape curve. We assume that $\mathbb{W} < \mathbb{S}$.

Consider the second scenario where one AUV is near the target location and the other is not. Let T_1 be the time it takes for the far AUV to get to the target location. For simplicity, we stop at $T_1 \leq 2\mathbb{W} + \mathbb{S}$.

$$Cost = \begin{cases} \mathbb{W} + \mathbb{S} & T_1 \leq \mathbb{W} \quad (6.4a) \\ T_1 + \mathbb{S} & \mathbb{W} < T_1 \leq 2\mathbb{W} \quad (6.4b) \\ T_1 + 2\mathbb{S} + \mathbb{W} & 2\mathbb{W} < T_1 \leq 2\mathbb{W} + \mathbb{S} \quad (6.4c) \end{cases}$$

There is a large initial cost for a higher wifi wait time in Equation 6.4a. Once T_1 is larger than $\mathbb{W}$, the cost increases with T_1 until it passes $2\mathbb{W}$ where the waiting AUV must then call the satellite for the second time, greatly increasing the cost. With a small ocean variance, the cost is mostly attributed to the first case.

Consider the third scenario where both AUVs are far away from the target location. Let T_1 and T_2 be the times it takes for the AUVs to get to the target location. Let $T_d = |T_1 - T_2|$ and

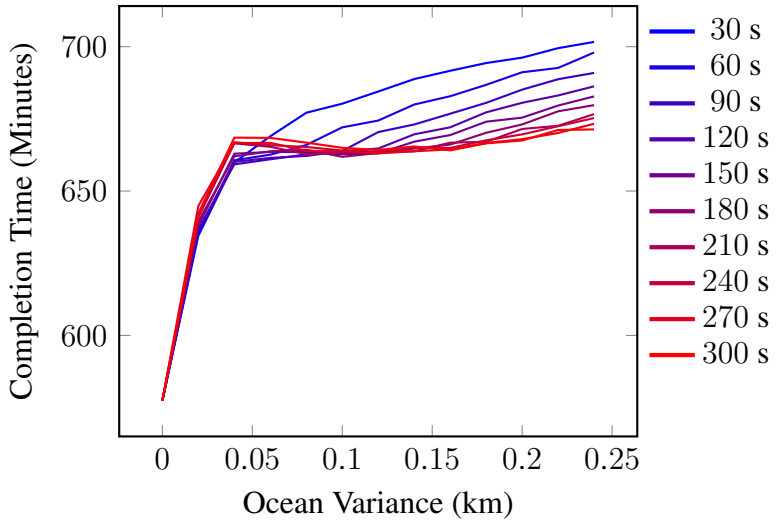

Figure 6.10: The relationship between team plan completion time, the variance of the ocean currents, and the wifi wait time for R-SATELLITE. The legend is wifi wait time in seconds. See Section 6.3.4.

$T_m = Max(T_1, T_2)$. For simplicity, we stop at $T_d \leq 2\mathbb{W} + \mathbb{S}$.

$$Cost = \begin{cases} T_m + \mathbb{S} & T_d \leq \mathbb{W} \quad (6.5a) \\ T_m + 2\mathbb{S} + \mathbb{W} & \mathbb{W} < T_d \leq 2\mathbb{W} + \mathbb{S} \quad (6.5b) \end{cases}$$

In this scenario, $\mathbb{W}$ does not increase the cost for the first case (6.5a), while at the same time, increasing $\mathbb{W}$ does increase the time threshold before the AUV jumps into the more costly case (6.5b), which is beneficial. With these equations, we can give a simple intuitive example on why the u-shape occurs.

Let $\mathbb{S}$ be 10 minutes and $\mathbb{W}$ be 5 minutes. For the second scenario, let T_1 be 2 minutes. The cost is then $\mathbb{W} + \mathbb{S} = 15$ minutes. For the third scenario, let T_1 be 3 minutes and T_2 be 4 minutes. The cost is then $Max(T_1, T_2) + \mathbb{S} = 14$ minutes. Therefore, even though the AUVs are farther out in the third scenario, the total cost for reuniting is less. The additional cost for the second scenario is because the AUV at the target location wastes time trying to connect and then eventually connects over satellite, which has a strict cost of 10 minutes.

6.3.5 Wifi range

The range of the wifi signal can have implications on the usability of intra-robot replanning with AUVs. We have assumed that there exists a wifi signal with enough range that the AUVs can communicate locally. However, the range of the signal may not be that far or non-existent at all, e.g., on a very turbulent ocean surface.

In Figure 6.11, we show multiple experiments testing different wifi range values for all the replan policies. The wifi range went from 0 to 0.3 km incrementing by 0.02 km. For ocean currents, we have a normal distribution on the x-axis with mean 1 and variance 0.05 and no y-axis current. Each replan policy is run 400 times for each wifi range.

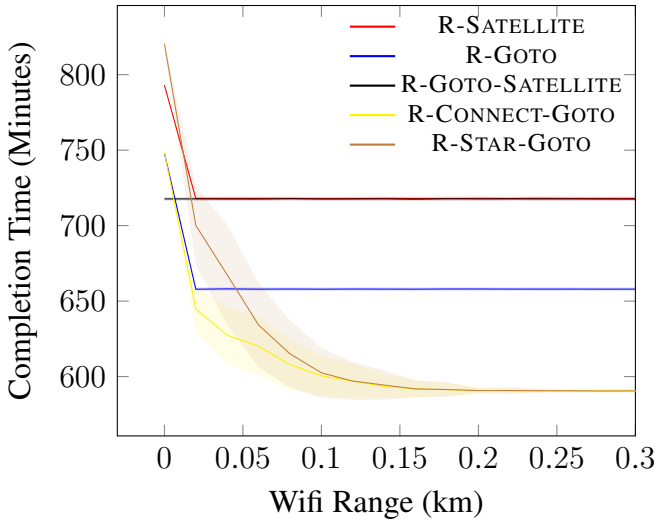

Figure 6.11: The relationship between team plan completion time and the wifi range for each replan policy. Highlighted area shows 1σ. See Section 6.3.5.

Notably, R-GOTO-SATELLITE performs the best when the wifi range is zero. This is because it goes immediately to the target location and then calls the satellite so that the AUVs sync over the satellite without the need for wifi. On the other hand, R-SATELLITE performs worse because in the worst case the AUVs call the satellite twice, first at the failed location and then at the target location to sync. R-STAR-GOTO performs the worst because it wastes so much time attempting to connect over wifi. As the wifi range expands, the improvements of R-GOTO, R-CONNECT-GOTO, and R-STAR-GOTO can be seen. Therefore, it is important to understand the wifi capabilities of the AUVs using intra-robot replanning and the effects the wifi range has on the performance of an individual replan policy.

6.4 Summary

In this chapter, we show that intra-robot replanning can be very beneficial when considering replanning for multiple AUVs in the ocean. We show how replanning locally rather than calling the satellite for every failure can greatly reduce the total time it takes for the AUVs to complete their assigned team plan. We then demonstrate how different variables can have profound effects on the performance of intra-robot replanning. The cost for communicating over satellite, the ocean current variance, the maximum wait time for connecting over time, and the wifi range can change the performance of individual replan policies, so the cost function for selecting replan policies should attempt to take these variables into account if possible. Another option is to learn the cost of each replan policy. Gathering enough data to properly learn the cost functions for each replan policy can be arduous. On the other hand, with the rise in long-term autonomy, i.e., robots running for months to years at a time, there may be enough time to let the AUVs gather data when using different replan policies and to slowly learn an effective cost function.

Chapter 7

Grid Domain

In this chapter, we explore the benefits of the rationale-driven plan with intra-robot replanning in a targeted planning domain. We run multiple experiments testing different aspects of the problem of replanning locally. In doing so, we attempt to highlight the key benefits of our approach and when intra-robot replanning with a rationale-driven plan should be used. We first detail the Planning Domain Definition Language (PDDL) domain that we use for our testing [Ghallab et al., 1998]. Then, we describe the planning algorithm we use for generating the rationale-driven plan. Lastly, we investigate multiple experiments and the replan policies we use within them to demonstrate the benefits of a rationale-driven plan.

7.1 Domain

The domain we use for our experiments is the Grid domain originally used in the AIPS-98 competition [McDermott, 2000]. A visualization of an example problem we use in our experiments is shown in Figure 7.1. A grid of rooms is kept at a size of 5 by 5 for a total of 25 rooms for our experiments. Doors are placed between rooms with a connection, (conn ?x ?y) in the PDDL. Within the rooms, there can be various keys with different shapes on the ground. Some of the rooms are locked (room outlined in red in figure) and they have designated lock shapes (red shape in the bottom left corner) that can be unlocked with the corresponding key. The goal(s) in this domain are to place one or more keys at various locations in the grid. The robot's actions are: (i) to move from its current location to a neighboring room, (ii) pick up a key if the robot and the key are in the same room, (iii) unlock a door if the robot is holding a key with the same shape, (iv) pick up a new key and drop (or lose) its currently held key, and (v) put down the key it is currently holding. The full PDDL description for the domain is shown in Figure 7.2.

7.2 Planning Algorithm

For this chapter, we use a forward search using the A* algorithm, [Hart et al., 1968], along with the Fast-Forward heuristic, described in [Hoffmann and Nebel, 2001], for the estimated goal cost. Depending on the planner, there can be variable amounts of information available in the final plan. However, the constraints and why the action was chosen should be available across different planners. These are vital to determining if a failure in the planner's assumptions has occurred.

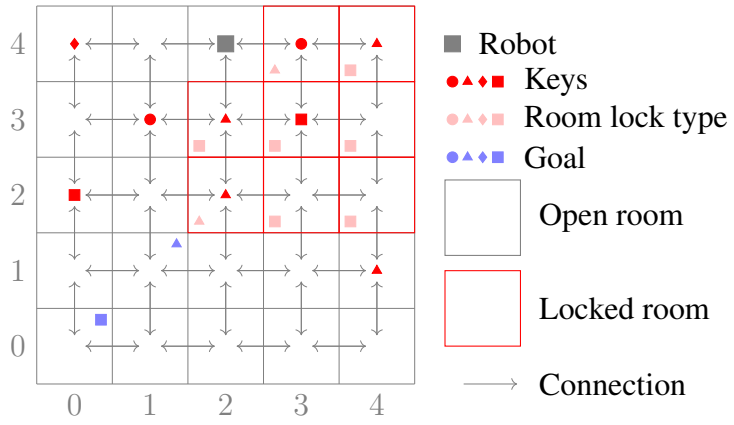

Figure 7.1: Grid domain example visualization. Solid gray box is the robot, red shapes in the centers are keys, shapes in the bottom left corners are lock types, rooms outlined in red are locked, blue shapes in the upper right corners are goals, and arrows are allowed movements.

```
(define (domain grid) (:requirements :strips)
(:predicates (conn ?x ?y) (key-shape ?k ?s) (lock-shape ?x ?s) (at ?r ?x ) (
    at-robot ?x) (place ?p) (key ?k) (shape ?s) (locked ?x) (holding ?k) (open
     ?x) (arm-empty ))
(:action unlock :parameters (?curpos ?lockpos ?key ?shape)
:precondition (and (place ?curpos) (place ?lockpos) (key ?key) (shape ?shape)
    (conn ?curpos ?lockpos) (key-shape ?key ?shape) (lock-shape ?lockpos ?
    shape) (at-robot ?curpos) (locked ?lockpos) (holding ?key))
:effect (and (open ?lockpos) (not (locked ?lockpos))))
(:action move :parameters (?curpos ?nextpos)
:precondition (and (place ?curpos) (place ?nextpos) (at-robot ?curpos) (conn ?
    curpos ?nextpos) (open ?nextpos))
:effect (and (at-robot ?nextpos) (not (at-robot ?curpos))))
(:action pickup :parameters (?curpos ?key)
:precondition (and (place ?curpos) (key ?key) (at-robot ?curpos) (at ?key ?
    curpos) (arm-empty ))
:effect (and (holding ?key) (not (at ?key ?curpos)) (not (arm-empty ))))
(:action pickup-and-loose :parameters (?curpos ?newkey ?oldkey)
:precondition (and (place ?curpos) (key ?newkey) (key ?oldkey) (at-robot ?
    curpos) (holding ?oldkey) (at ?newkey ?curpos))
:effect (and (holding ?newkey) (at ?oldkey ?curpos) (not (holding ?oldkey)) (
    not (at ?newkey ?curpos))))
(:action putdown :parameters (?curpos ?key)
:precondition (and (place ?curpos) (key ?key) (at-robot ?curpos) (holding ?key
    ))
:effect (and (arm-empty) (at ?key ?curpos) (not (holding ?key)))))
```

Figure 7.2: Grid PDDL domain where a robot can move around, pick up and drop keys, and unlock rooms [Ghallab et al., 1998].

The preconditions and effects are added when the action is generated for a particular state in the forward search. The parameters of the preconditions and effects are picked by applicability in the state so there is no extra rationale for picking them. We explore extra rationales for the parameters in Experiment 3. When the forward search finds the goal state, it generates the plan going from the goal state back to the initial state. Important for this domain, we add the predicates that need to remain valid, i.e., persistent conditions, as constraints to the action during this phase of the plan generation, see Algorithm 7.1. Specifically, predicates like *holding* are added as constraints to the *move* action if a key will eventually be needed to unlock a door. The assumption of the planner is that nothing outside of the effects of actions changes the environment, therefore, there was never a requirement to maintain an understanding that the planner assumed previously achieved predicates would remain valid. However, in dynamic environments this assumption needs to be addressed by adding them as constraints to the action. Also recall that in Algorithm 3.3, the planner labels effects superfluous if they do not contribute to the outcome of the plan, meaning for all intents and purposes the effect has been removed.

For our example, we add the persistent constraints and remove the superfluous effects with Algorithm 7.1. The algorithm requires the plan, which is a list of actions starting with a null action, and the goals. On line 1, the current action index j is set to equal i, the index of the last action in the plan. A set of constraints is initialized on line 2. Then the algorithm starts the loop that will loop over all the actions on line 3. For every effect of a_j, line 4, if the effect is not an element of the union of the goals or the constraints, line 5, then the effect is removed as superfluous, line 6. If the effect is an element of the set of constraints, line 8, then the effect is removed from the set of constraints, line 9. After all the effects are checked, the current set of constraints is added to a_j, line 12. Then the preconditions of the current action, a_j, are added to the set of constraints, line 13. The current action index, j, is subtracted by 1 thereby updating the current action, line 14. Finally, if the current action, a_j, is equal to $\emptyset$, then the loop ends, line 15.

In Figure 7.3, we highlight two plans, one including only the preconditions and effects and the other, our rationale-driven plan, using Algorithm 7.1. We specifically focus on the condition **holding** and highlight the difference between the two plans when considering that constraint. We can see that when going through the plan backwards that the condition **holding** is added as a constraint between an action with the precondition of **holding** and an earlier action with the effect of **holding**. Note that we simplified the example by not including all of the preconditions, effects, and constraints that would normally have been included in the example plans. The **holding** constraint is a key addition to the rationale-driven plan in this domain and the addition of this constraint improves the performance of our individual robot using our intra-robot replanning algorithm.

In regard to intra-robot replanning, the rationales are sorted by the ordering: *constraints*, *why-this-action*, *preconditions*, and then *effects*. There are never multiple policies that handle the same rationale in this chapter's experiments so we do not sort them; the previous Chapters 5 & 6 explore this area.

7.3 Experiment 1: Failures in the Grid Domain

We generate ten different problems in the Grid domain, each with one or two goals. For this experiment, the cost of each action is 1 while the cost of communicating with the centralized planner is 10. During the execution of each action in the plan, there is a 30% chance of failing to

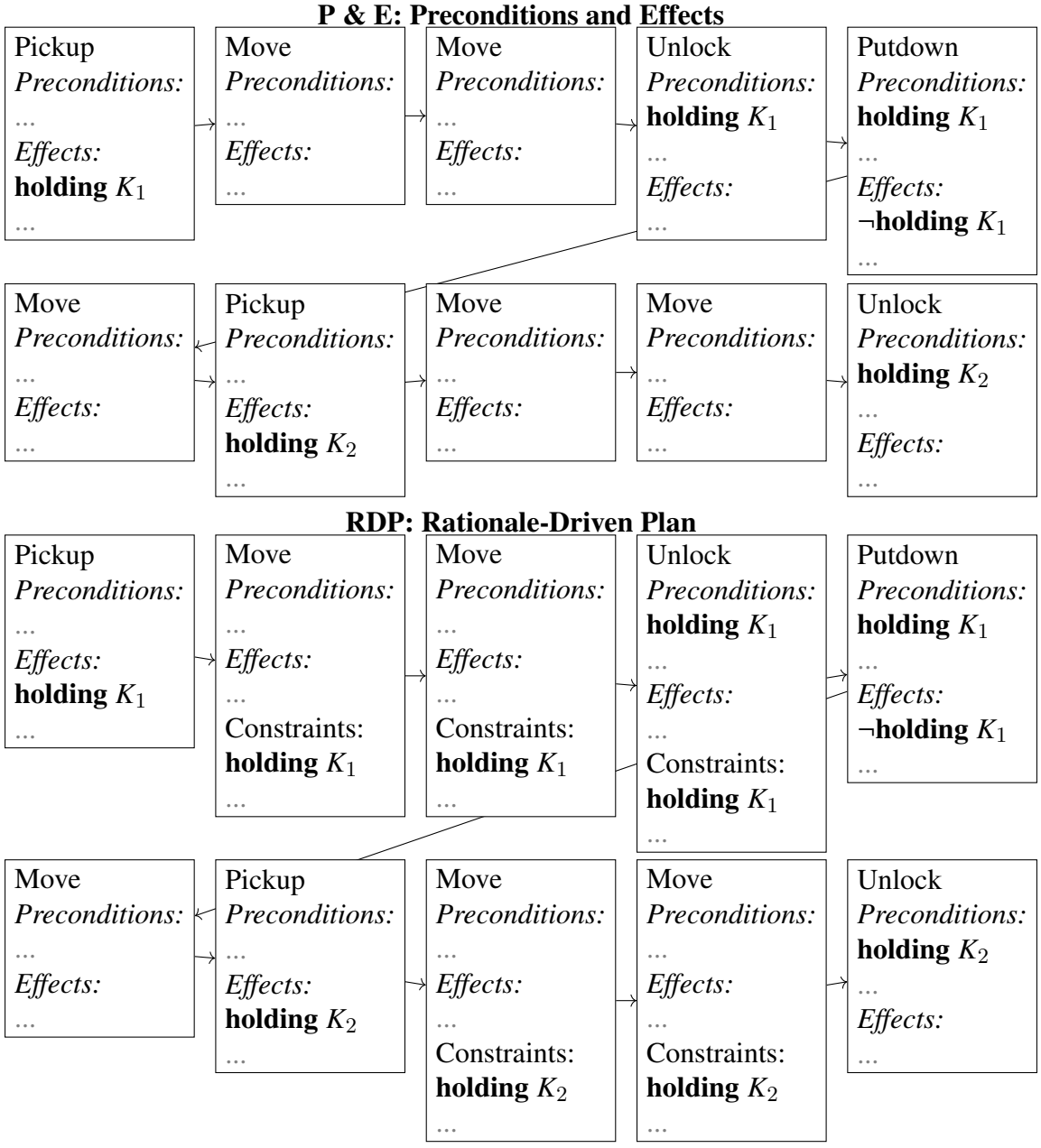

Figure 7.3: The example highlights the difference in the **holding** condition between the **P & E** plan and the **RDP** plan. Specifically, the **holding** condition is added as a constraint to some actions in the **RDP** plan based on Algorithm 7.1.

Algorithm 7.1 Algorithm that adds the constraints that must remain valid during execution to actions in the plan. This is a post-processing of the plan that requires the plan, P, that contains a list of pointers to the actions of the plan and the set of goals, G.

Require: $P = \{a_0 = \emptyset, a_1, a_2, \ldots, a_i\}$, G
1: $j = i$
2: C = $\{\emptyset\}$
3: **repeat**
4: **for all** $e \in a_j$.effects **do**
5: **if** $e \notin$ G $\cup$ C **then**
6: a_j.effects = a_j.effects $\setminus\{e\}$
7: **end if**
8: **if** $e \in$ C **then**
9: C = C $\setminus \{e\}$
10: **end if**
11: **end for**
12: a_j.constraints = C
13: C = C $\cup$ a_j.preconditions
14: $j = j - 1$
15: **until** $a_j \equiv \emptyset$

move (robot remains in its current location) and of dropping a key, if a key is held. If the key is dropped while the robot is moving, then the key is placed in the previous location of the robot, i.e., the robot left the key behind before moving.

7.3.1 Replan policies

1. *Replan-Move* $\langle \beta = \textit{at-robot} \rangle$: Attempts to move directly to the failed location from its current location.

2. *Replan-Pickup* $\langle \beta = \textit{holding} \rangle$: If the key is at the robot's current location it picks the key up, otherwise it attempts to move directly to the key's location. If the robot fails to move (no connection between rooms) or fails to pick up the key once, then it fails.

The results for Experiment 1 are shown in Figure 7.4. The *Plan Length* is the original length of the plan provided by the planner. Therefore if there were no failures, the cost of the plan should be equal to the length of the original plan. The closer the robot's final cost gets to the original length, the better the robot is at handling failures. The performance of any approach will never reach the *Plan Length* because of the high failure rate in this experiment.

For *Centralized*, the robot is provided the preconditions and effects, but only to detect immediate failures in order to call the centralized planner for a new plan. *Centralized* clearly stands out as the most expensive approach. The failure rate in this experiment is relatively high so the cost of communicating adds up quickly.

For *P&E* (Preconditions and Effects), the robot is provided the preconditions and effects along with the replan policies, previously described. *P&E* attempts to repair the plan using the replan

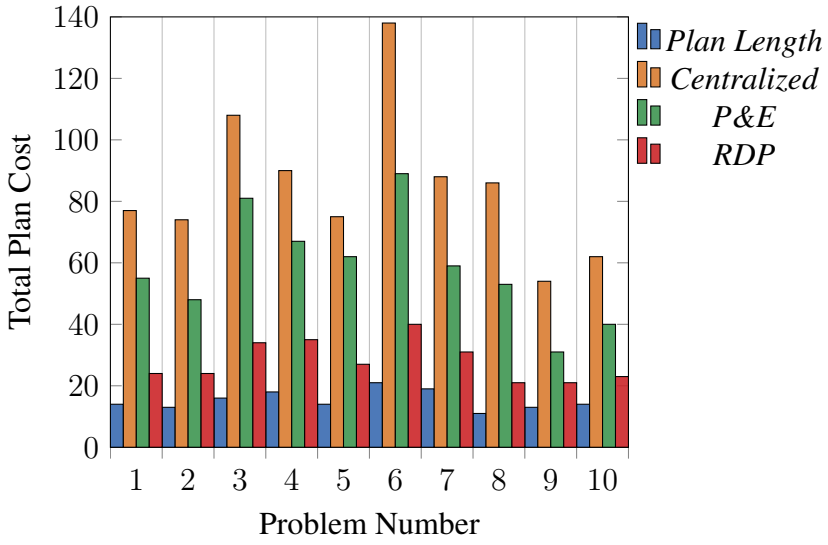

Figure 7.4: Compares different approaches of replanning to the plan length which is the optimal case when no actions fail, see Experiment 1. Values are an average of 100 runs.

policies, but some failures like dropping a key that is needed later in the plan are realized too late and ends up making *P&E* call the centralized planner. *P&E* does better than *Centralized* but the total cost is still rather expensive.

For *RDP* (Rationale-Driven Plan), the robot receives the rationale-driven plan and the replan policies. Clearly *RDP* performs the best at solving the failures locally. The major difference between *P&E* and *RDP* is that while moving, *P&E* may lose a key and then after finally getting to the locked door realize that the holding predicate is false. In this case, *P&E* must call the centralized planner since the key is too far away for *Replan-Pickup* to work. *RDP* is provided with the constraint that the predicate *holding* must remain true while it moves to unlock or drop off a key, and so it can replan and pick up the key immediately after it has fallen from the robot's grip. Ultimately this results in *RDP* rarely calling the centralized planner for a failure.

7.4 Experiment 2: The Effect of the Communication Cost

Next, we want to understand the effect of different communication costs on the performance of each approach. We vary the cost from 0 to 20, shown in Figure 7.5. These results are generated using problem number 1, from Figure 7.4. Interestingly, the rationale-driven plan can be beneficial even with low communication costs.

When the communication cost is 0, the three approaches' total costs are based purely on the number of actions needed to reach the goal. Their ordering at 0 cost may seem surprising at first because *RDP* still outperforms *Centralized*. We must first understand that *P&E* performs the worst because when it drops a key and leaves it behind, it may attempt an infeasible *move* action trying to get back to the key's location, thereby wasting time rather than immediately calling the centralized planner. *Centralized* does not attempt infeasible actions, but it still acquires the cost associated with going back to pick up the key. On the other hand, *RDP* recognizes immediately that it dropped

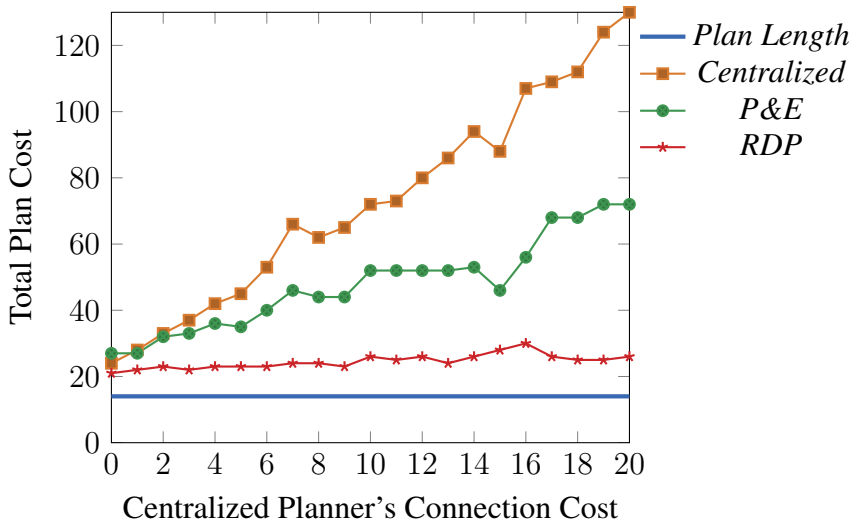

Figure 7.5: Compares different replan policies, for problem number 1, as the cost of communicating with the centralized planner increases, see Experiment 2. Values are an average of 100 runs.

a required key and so it replans earlier. This specific rationale added to the rationale-driven plan allows *RDP* to perform best, in total plan cost, even when the cost of communicating with the centralized planner is 0.

As the cost increases, the performance of *P&E* matches that of *Centralized* for a short while as the cost of calling the centralized planner matches the cost of sometimes failing to replan. However, *Centralized*'s cost increases quickly as the communication cost increases beyond 4. *P&E* does slightly better by handling immediate failures, like failing to *move*, without needing to add the communication cost. *RDP* maintains the lowest cost because of its unique ability to handle almost all the failures locally, so it rarely needs to call the centralized planner.

These results demonstrate that as the communication cost with the centralized planner increases, implementing intra-robot replanning with a rationale-driven plan becomes more and more cost-effective. The noticeably more striking point is that *RDP* performed better than *Centralized* even when the communication cost is 0 for this problem.

7.5 Experiment 3: Benefits while Communication Cost is Zero

In the previous experiment, we demonstrate for a single problem that *RDP* performed better than *Centralized* when the communication cost is 0. So for this experiment, we test every problem from Experiment 1 with communication cost 0 to see if this occurs in other problems. Additionally, we make the environment more difficult by adding the possibility of failure when the robot places a key down (*putdown* or *pickup-and-loose*). Specifically, there is a 60% chance that the key is not placed at the correct location. Of these failures, 50% are placed *near* and 50% are placed *far* from the location. In this case, the effect of the action is invalid and the robot needs to replan accordingly. Also, two open connections to and from the new key location to the robot's current

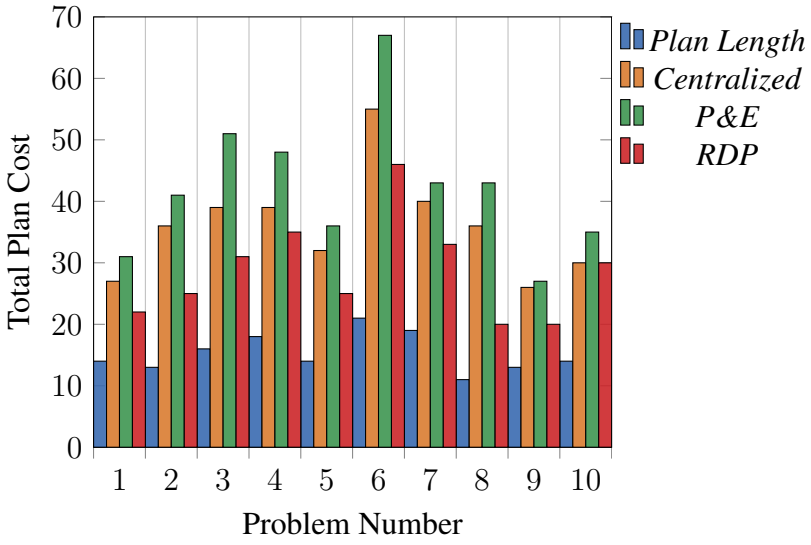

Figure 7.6: Compares different approaches of replanning to the plan length which is the optimal case when no actions fail, see Experiment 3. Values are an average of 100 runs.

location are added to the state after such a failure.

For our rationale-driven plan, we add the constraint *near* to all actions involving putting down a key. Specifically, the constraint defines that the key must be at the correct location or the *near* location. To handle the *at* failure, we introduce the replan policy:

- *Replan-At* $\langle \beta = at, C = near \rangle$: Attempts to move to the key location, pick it up, move back, and drop the key. If *near* is provided and the key is near the location, then it replaces the action's effect and solves the failure.

In PDDL, the goal could be modified to:

```
(or (at key place) (at key near-place))
```

and then *relevant-to* would point to both goals. As such, the replan policy would need to be slightly modified. However, the places in the state would triple and greatly increase the search problem for the planner. We keep the problem simple, but in doing so the centralized planner always plans for the exact location. We also do not take into account the time needed for the centralized planner to replan, and depending on the planner this could make *Centralized* even worse in total plan cost.

Shown in Figure 7.6 are the results of this experiment. Clearly, the performance depends on the problem, however for almost every problem, *RDP* has the lowest total plan cost. *P&E* has the worst cost because it fails sub-optimally, i.e., too late after losing a key, and because using the replan policies can be sub-optimal to the overall objectives of the centralized planner. *Centralized* also fails sub-optimally but it can replan with its own objectives and not waste time with sub-optimal actions. *Centralized* is also slightly penalized because it requires the robot to place the key on the exact spot whereas *RDP* can place it near the location. These results demonstrate that *RDP* continues to outperform *Centralized* across most problems that we test and highlights the effectiveness of the rationale-driven plan.

7.6 Summary

In this chapter, we present multiple experiments within the grid domain that highlight the benefits of intra-robot replanning and the rationale-driven plan. We use a forward search method to solve the planning problem, and we generate the rationale-driven plan during the planning process. The rationale-driven plan has a plethora of information about the centralized planner's reasoning that a typical plan would leave out. The added rationale facilitates intra-robot replanning without the costly communication with the centralized planner. The more the cost of communicating with the centralized planner rises, the greater the potential benefit. However, we demonstrate that even with a very low cost there is a benefit for using intra-robot replanning with a rationale-driven plan.

Chapter 8

Learning

In this chapter, we describe our method for choosing a score (or metric) for each replan policy used in the intra-robot replanning algorithm described in Chapter 4. The scores are used to sort the replan policies in the intra-robot replanning algorithm, and the final ordering can have a significant effect on the overall performance. First, we describe an approach using the *worst case* analysis for choosing the appropriate replan policy. Second, we describe a *state based* approach that learns which replan policy to use given the robot's current state of the environment. This approach allows the robot to sort and ultimately select the best replan policy based on current environment variables. We then describe some results in learning the predicted probabilities of success in the robot soccer domain and in learning the predicted costs in the autonomous underwater vehicle domain.

8.1 Worst Case

Recall the robot soccer domain previously described in Chapter 5. The task of our robot is to pass the ball to another teammate. This involves releasing the ball and losing all further control over it. It is an irreversible action, which cannot be replanned or halted. With irreversible actions, we prefer to choose a replan policy that minimizes the worst case failure rate of the robot, because most of the failures cause the other team to obtain the ball.

Provided there is a problem, the individual robot must sort the replan policies and select one to execute in order to accomplish the task of passing. We define a method of choosing an algorithm based on the worst case. A standard approach for choosing one algorithm over another is if the algorithm is better in the worst case independently of the worst case actually occurring. Let $Z = \{0, 1\}$, where 0 is a failure and 1 is a success, and p be the function that gives the probability $p(z)$, where $\sum_{z \in Z} p(z) = 1$, for obtaining the failure or success in Z. We then assign a value $v(z)$ to the values of Z. We set $v(z)$ to be the total number of successes for the algorithm, in Figure 5.6. We use the definition in [Rubinstein, 2006],

$$p \succsim q \text{ if } min\{v(z)|p(z) > 0\} \geq min\{v(z)|q(z) > 0\} \tag{8.1}$$

Therefore, the algorithms are ordered $C_Z \succsim D_M^1 \succsim D_M^0 \succsim BaseCase$. So, the **Change-Zone** algorithm has the largest minimum value and is chosen as the default algorithm.

8.2 State Based

The worst case method chooses the algorithm that minimizes the worst case scenario, but by definition it does not consider the actual probability of the worst case happening. This can be seen as an issue if the worst case rarely, if ever, happens. Another possible issue is that each of the replanning algorithms alters the team plan causing changes that may be unnecessary. In a normal robot soccer game, situations requiring complex replanning policies may not occur with high probability. In other words, if there is no longer an open pass to our teammate but there are no opponents nearby, then there would be no point in dribbling the ball and/or moving the teammate closer.

We argue that a better method of picking a replan policy is to base this decision on the state of the world. Using the state information, it is possible to learn the likelihood that a replan policy will be successful or to learn the estimated cost that the replan policy will have. Then we can order the replan policies based on the learned score, e.g., likelihood of succeeding. We must reiterate that because the domain is dynamic and adversarial, the probability of success or the estimated cost varies even in the exact same state. For example, the soccer robots move with some randomness which means that predicting the likelihood of an algorithm being successful at any given state is probabilistic. The soccer robots likely diverge quickly into very different states in the future as subtle differences and randomness influence their trajectories.

8.3 Learning Predicted Probabilities of Success

In this section, we detail our learning approach to sorting the replan policies in the robot soccer domain. For each replan policy, we train a neural network to learn a function from the environment state to the probability of success. The state is described in the reference frame of the robot using the algorithm in order to help generalize different situations across the field.

Input State

- T_b: (X, Y) position and velocity of T_b.
- **Ball**: (X, Y) position and velocity of ball in robot's frame of reference.
- **Teammates**: (X, Y) position and velocity of each teammate in robot's frame of reference.
- **Opponents**: (X, Y) position and velocity of each opponent in robot's frame of reference.

Output State

- **Probability Value**: Likelihood of input state leading to a successful pass using an algorithm.

The data needed to train the neural network is collected over many runs of each algorithm in various scenarios. We simulate similar episodes in our example domain with variations on ball location, opponent's locations, and teammate's locations. As an episode starts, we begin saving state information for each video frame. Then once the pass is either received, intercepted, or out of bounds (the episode ends), we label all the saved states with a success value of 1 or a failure value of 0. As previously mentioned, even starting in the same state does not mean the robots execute

the same path in the future even when using the same algorithm. There is therefore a probabilistic nature to succeeding from a given state.

The neural network is then trained using a supervised learning method. Given that similar, even the exact same, states are most likely labeled both success and failure, the network actually learns the probability of succeeding from that particular state within the training data. Therefore, the output is the likelihood of an algorithm succeeding from a given state.

Given that we train multiple neural networks, each trained on one replan policy, the intra-robot replanning algorithm has the likelihood of success for each replan policy given a state. These probabilities are then used to sort the replan policies so that the individual robot selects the one with the highest probability.

8.3.1 Experimental Evidence

To test the state based method, we use the open source library OpenANN [Fabisch, 2017]. The neural network has 20 inputs, see *Input State* above, and 1 output as the probability of success, see *Output State* above. We use a fully-connected network with three hidden layers with 100 neurons in each layer. We use LOGISTIC activators for the neurons and Mini-Batch Stochastic Gradient Descent (MBSGD) to solve the supervised learning problem.

Our work focuses on determining the accuracy of predicting the success or failure of an algorithm using our example domain. We use the D^1_M algorithm and repeatedly run our example domain roughly five thousand times, gathering over 4 million states. We use two hundred thousand states to train the network over ten thousand iterations of MBSGD.

In Figure 8.1, we see that the accuracy of prediction is between $0.65 - 0.71$. The low accuracy can be attributed to the randomness of a state leading to a success or failure because of the inherit randomness in the robot's performance. For example, the starting position of our *passing with marking* domain should be labeled as failure because on average it fails more often than it succeeds. This is confirmed as the start position produces the value 0.412. The neural network always considers that position as a failure, which brings down the accuracy when the robot happens to succeed. Recall that in Chapter 5, the replan policy Dribbling-Move (D^1_M) had a success rate of 0.37 in Figure 5.6. Here, the neural network learns a similar approximation.

The low recall performance can be attributed to the large bias for failure in the dataset. The individual robot is started in a disadvantaged position and the likelihood of succeeding is already low. Similarly, when the robot actually succeeds in a given state it has probably failed multiple times from that exact same state in the dataset. The negative examples (failures) then highly outnumber the positive examples (successes) leading the neural network to perform poorly on

Predicting Success or Failure for D^1_M			
Data Sets	**Accuracy**	**Recall**	**Precision**
Training Data	0.70	0.31	0.74
Next 200,000	0.65	0.26	0.65
Next 200,000	0.71	0.24	0.50

Figure 8.1: Accuracy of predicting successes or failures for our example domain using the algorithm D^1_M.

recalling positive examples. A possible correction for this poor performance is to balance the number of positive and negative samples in the training set, however, this new bias would likely decrease the precision and accuracy of predicting if the algorithm will fail from a given state.

This work demonstrates the ability to learn the probability of success for a given state in the robot soccer domain.

8.4 Learning Predicted Costs

In this section, we detail our learning approach to sorting the replan policies in our AUV domain. We use the state based approach that learns a predicted cost for each replan policy given the current state of the environment. We then demonstrate that learning the predicted costs can improve team performance compared to a deterministic approach in Section 8.4.1.

We train the neural networks to learn a function from a state of the environment to a predicted cost. The predicted cost is the time it takes for the replan policy to complete (successfully or unsuccessfully) plus the time it takes to complete the next action in the team plan. Our predicted cost is therefore a two step lookahead, i.e., what it costs to use this replan policy and what the next action costs. For the AUV domain, we use the following:

Input State

- AUV's current location (X,Y)
- AUV's destination location ($\mathbb{X}$, $\mathbb{Y}$)

Output

- Predicted cost of replan policy and the next action

The data to train the neural networks is collected over multiple runs of the simulation. For each replan policy, we order it first and run the entire team plan. We save the current state when replanning starts, along with the cost it takes to replan plus the cost of the next action. Each AUV collects its own state and cost during the execution of the team plan. A neural network is then trained for each AUV for that particular replan policy using a supervised learning method. We repeat until we have trained a neural network for every replan policy.

For the learning, we use the open source library OpenANN [Fabisch, 2017]. For each replan policy, we use a full-connected network with one hidden layer with 50 neurons. We use RECTIFIER activators for the hidden layer and a LINEAR activator for the output. In the experimental environment, we run each replan policy (ordered first) 5,000 times collecting training data. The networks are trained using OpenANN's built-in conjugate gradient method (max 5,000 iterations, min error 10^{-8}, mean squared error, and 35,000 samples each). The neural networks are then used to sort the replan policies before execution by LEARNED.

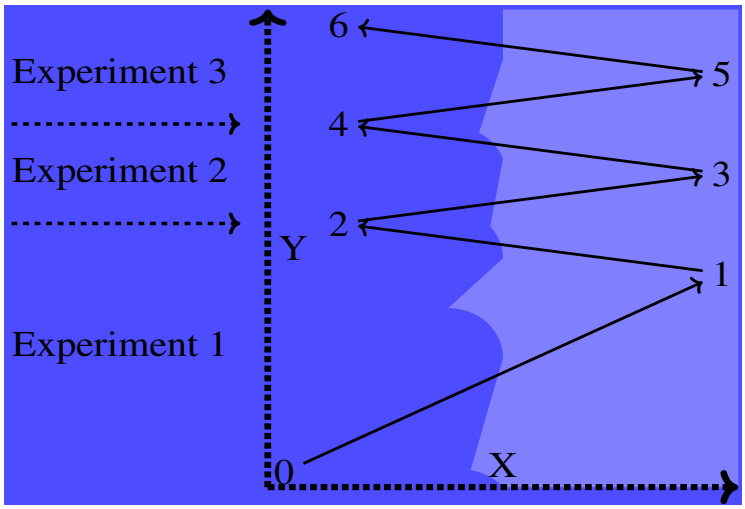

Figure 8.2: A simulated environment where two AUVs criss-cross the boundary line of fresh water (light blue) and salt water (dark blue) to gather scientific data about the boundary. The ocean is divided along the y-axis into areas with different ocean currents, and the type of current corresponds to the experiment number. See Section 8.4.1.

8.4.1 Learning Costs for Autonomous Underwater Vehicle Domain

We combine the ocean currents of the autonomous underwater vehicle experiments in Chapter 6 separated by the locations shown in Figure 8.2:

- Locations 1 & 2 are *Experiment 1*: We assume there is an ocean current in the positive y-axis modeled by a normal distribution with mean 2 and variance 0.1, and a uniform distribution from 0 to 0.1 in the x-axis.

- Locations 3 & 4 are *Experiment 2*: We assume there is an ocean current in the positive x-axis modeled by a normal distribution with mean 1 and variance 0.05, and no current (0) in the y-axis.

- Locations 5 & 6 are *Experiment 3*: We assume there is an ocean current in the positive x-axis and y-axis both modeled by a normal distribution with mean 1 and variance 0.2. We are assuming the surface waves are so large that connection over wifi **cannot** be established (no wifi connection).

Figure 8.3 shows that the LEARNED method (a learned ordering of the replan policies) outperforms a predetermined order of the replan policies in this complex environment. The improvement is made on the last two locations, 5 & 6, where there is no wifi connection. The predicted cost of using R-GOTO-SATELLITE is less than that of any other replan policy for those two locations. This saves five minutes of wait time at each location, wasted by R-CONNECT-GOTO when it attempts to connect over wifi but fails. Notably, we see a clear improvement of roughly ten minutes for the LEARNED method. In this way, LEARNED effectively uses R-GOTO-SATELLITE only when it is the best option given the state of the environment.

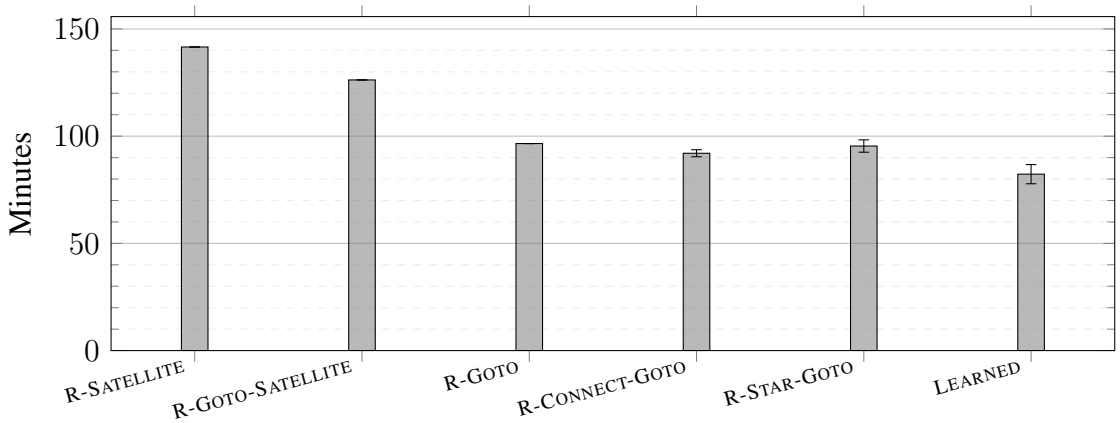

Figure 8.3: *Experiment, changing ocean currents for different locations*: Average time and variance to complete the team plan 5,000 times using the stated replan policy. LEARNED performs the best in this complex domain by combining the benefits of R-CONNECT-GOTO and R-GOTO-SATELLITE.

8.5 Summary

In this chapter, we present our state based method for learning a score for each replan policy used by the intra-robot replanning algorithm. The score is then used to sort the replan policies and to improve the performance of the individual robot and the team. We present experimental results in learning the predicted probability of success in the robot soccer domain and in learning the predicted cost in the autonomous underwater vehicle domain.

In the next chapter, we discuss the related work to this thesis. Research is a process of building on and adding to the research of others, and as such there has been plenty of research that has inspired the approaches in this thesis. There are also many avenues for future work within the intra-robot replanning problem that we consider interesting and worth pursuing, which can be found in Chapter 10.

Chapter 9

Related Work

In this chapter, we describe the related work on: team plan representations; execution, failures, and replanning; and learning. The various team plan representations describe different approaches to solving the multi-robot planning problem and the different considerations they take when creating the plan's representation. One such key consideration is the trade-off between using local information and global information when solving the multi-robot problems [Parker, 1993]. In addition, recent work has explored the formal conditions for required cooperation to solve a multi-robot problem [Zhang et al., 2016]. In dynamic environments, there is no plan representation that can guarantee zero failures, and so the next section focuses on the important step of actually executing a plan and handling possible failure cases. Lastly, we explore some literature regarding machine learning for multi-robot problems, where our focus is on improving the robots' performance over time.

9.1 Team Plan Representations

9.1.1 STEAM

STEAM, [Tambe, 1997], is a general model for teamwork and communication that provides the team members with a way to reason about coordination and communication when completing their tasks autonomously. STEAM uses the *joint intentions* theory, [Levesque et al., 1990], for its teamwork and communication requirements. The joint intentions theory describes persistent shared commitments between agents where they must agree on their commitment to a task and agree on their dissolving of the commitment. The communication requirements are derived from a mutual belief that must be met for persistent shared commitments. The key assumption with this method is that team members will tell the truth and always attempt to share their intentions.

STEAM is used to execute hierarchical reactive plans and *team operators* (reactive team plans). Within the plans, if one of the actions requires the whole team then the whole team must create a joint intention to execute it. Similarly, if the action only requires a sub-team then that sub-team must create a joint intention. This way the team works towards its goal(s) using the same reactive solution, which given the joint intention theory is not a requirement. Individual agents can produce different solutions that may cancel each other out.

Referring to team operators, they prescribe to a classical planning approach with: (i) precondi-

tions rules, (ii) application rules, and (iii) termination rules. The particular instance of the operator, team or individual, is determined dynamically by STEAM by how many agents are executing the action.

STEAM also requires a certain level of communication between team members. They must share their joint intentions so that the other team members can infer what step the sub-team intends to execute. STEAM also checks that the canceling of any action does not affect the ability of another action to be accomplished, otherwise, the information is shared to that sub-team(s).

9.1.2 Skills, Tactics, and Plays (STP)

In [Browning et al., 2005], STP is a hierarchical structure that decomposes the multi-robot problem of team planning into Skills, Tactics, and Plays. Skills are the low level repeatable algorithms that accomplish a very specific, sometimes complex, task. Tactics then combine skills using a skills state machine (SSM) and define the transitions between each skill represented as the states within the SSM. The Plays defines the team plans or the joint policies for the entire team.

The Plays, similar to classical planning, define: applicability conditions, termination conditions, roles, and execution details. Applicability conditions are logical formulas based on the predicates of the world, i.e., a set of preconditions for the Plays to be applicable. Given the dynamic adversarial environment STP was designed to handle, the effects of the Plays cannot be known perfectly beforehand so termination conditions differ slightly from the conventional classical planning effects. Termination conditions list some of the possible outcomes and assign them a result (Success, Completed, Aborted, or Failure). The roles define temporally extended sequences of Tactics and their respectively defined parameters. Lastly, execution details define the play's timeout and, if applicable, opponent roles.

It is infeasible to generate all possible states and state transitions for many complex domains like robot soccer. Therefore, the STP solution is to generate multiple plays for certain conditions of the world, i.e., ball on offensive side. This creates a new problem of picking a play but it also demonstrates that there can be a mismatch between the applicability conditions and all the true conditions of the world. The termination conditions too might be missing some valuable termination criteria. This can lead to unknown failures for the team.

There has been recent work on extending STP to work in a decentralized method [de Koning et al., 2017]. Agreement on plays and roles are accomplished through a voting system that is communicated through shared information between robots.

9.1.3 Multi-Agent Simple Temporal Network (MASTN)

As defined in [Riley, 2005], MASTNs were introduced "as a plan representation for distributed execution" and they built upon the Simple Temporal Network formulation in [Dechter et al., 1991]. The author needed a multi-agent plan representation that could be constructed in a centralized manner and be executed by the agents in a distributed fashion. Therefore, MASTNs provided a representation that could encode enough information for the individual robots to coordinate with each other and to detect failures during execution.

More specifically, the formulation allows parallelism of actions and coordination between agents through the temporal constraints provided by the Simple Temporal Network. In addition, the nodes of the graph are events in the world but function similar to actions in classical planning.

They can be parameterized, and they include the agents that are involved in bringing about the events. The MASTN representation is a graph based multi-agent plan with temporal constraints for synchronizing actions between agents.

Recent work in temporal planning has considered the problem of multi-robot symbolic planning with temporal uncertainty [Zhang et al., 2017]. It brings together classical conditions and effects while including temporal uncertainty aspects to action completion and temporary effects.

9.1.4 Petri Nets

For Petri net team plan representation, we consider the research in Collaborative Agents for Simulating Teamwork (CAST) that uses Petri nets for their team's cognitive model [Yen et al., 2001]. The Petri net's framework relates to classical planning in: (i) the transitions can be considered actions, (ii) the inputs as preconditions, and (iii) the outputs as post-conditions. The authors then extend the model to have control nodes that maintain the beliefs about the current team goal(s) and the actions of others, and belief nodes that maintain the belief about the world. Monitoring and tracking the execution of the team is easily captured by the tokens that move through the Petri net.

The focus of this research is on anticipating the information needs of the other team members, i.e., a proactive approach to the information exchange process between individual agents. The information needs are gathered through the preconditions and post-conditions of the actions. Preconditions illuminate the information needed by an agent(s) to complete the action. Post-conditions produce new information and so can be used to proactively send information required by some precondition to that agent. It also checks for any new important information found that might be relevant to the preconditions of another agent. Before sending the information, the agent tries to figure out if the other agent already knows the information.

Recent work with Petri nets handles interruptions from outside sources, like human operators, within team plans [Farinelli et al., 2017]. It has shown a decrease in time to complete plans and decrease in the human operator load.

9.1.5 Market-Based Multi-Robot Coordination

Market-based multi-robot coordination covers a large area of research in coordinating teams or individual agents to accomplish tasks together, see review article [Dias et al., 2006]. First, there is an objective that can be broken down into sub-objectives or tasks. This component relies on some way to divide the objective, e.g., centralized algorithm or humans. In addition, the agents also have a finite set of resources that can be used. There is a global function that defines a value for all the possible solutions for the objective. Each agent has its own individual utility function that produces a value given the available resources to itself and its ability to contribute to the objective. Lastly, there is a mechanism for distributing the objective or sub-objectives, commonly auctioning.

In auctioning there is an auctioneer that takes the bids produced by the individuals during the bidding phase, and then it decides the winning bid and assigns the task to the winning individual. Prior to this step, the individuals often generate a plan for the task that is then used to produce a value, i.e., their bid. This solution to coordination can be interpreted as a mixture between fully centralized and fully distributed planning. It should be noted that in either of those extremes the market-based approach often performs worst due to the communication overhead (for distributed) or sub-optimal local planning (for centralized).

Ultimately, team planning in a market-based approach is distributed throughout the team members but there are some critical points to consider. The mechanism for sub-tasking the objective is very similar to the high levels of any hierarchical approach such that planning for the division of work must still be accomplished in a centralized way. The difference lies in that the market-based approach allows the sub-tasking algorithm to make no commitments on to *whom* the tasks are assigned. The assignment problem is pushed to the auctioneer and the individual robots' abilities to solve the sub-task (encompassed in their bids). The authors in [Schneider et al., 2014] demonstrate that there are trade-offs to consider for different auctioning algorithms based on optimality and the number of tasks to allocate. As is the general way of market-based coordination, the individuals are implicitly *self-interested* agents and this aspect hinders this method's general ability to handle tightly coordination tasks.

9.1.6 Decentralized Sparse Interaction Markov Decision Processes

Markov Decision Processes (MDPs), [Puterman, 1994], provide a way of modeling decision making in environments where the outcomes of actions can be random. MDPs are described with a finite set of states, a finite set of actions, a probability function that an action from some state will lead the agent to a certain new state, a reward function for transitioning into the new state, and a discount factor. Ultimately, the goal is to produce a policy that the agent can follow to maximize the reward function.

In the case of team planning, Multiagent Markov Decision Processes (MMDPs) are a generalization on MDPs to include multiple agents. MMDPs add a finite number of agents, add finite actions for each agent, change the probability function to be defined by the joint action space from one state to another, and change the reward function to be the expected reward received by all the agents taking the joint action. From a centralized planner's point of view solving a MMDP is no different from a MDP. In [Spaan et al., 2002], the authors included the notion of roles into their definition. Roles allowed for specific domain knowledge to be integrated into the MMDP and also reduce the complexity of the problem. Future work was left to actually learning the policies.

On the other hand, a Decentralized MMDP (Dec-MMDP) is where each agent observes local information but together they fully describe the state. This of course adds a problem of communication and the assumption that the state will be fully observable with all the local observations. There are some similarities between this direction of research and our own in considering the individual agent's (robot's) perspective within a team, but it does not join team planning and individual planning as policies are meant to be learned and solved for a problem. Replanning for all purposes is considered irrelevant however feasible that consideration might be in working with real robots. In general, solving Dec-MMDP problems is computationally expensive, NEXP-hard [Bernstein et al., 2000]. Therefore, there has been some work in improving the computational complexity by breaking apart the problem.

The [Melo and Veloso, 2011] assumption is that coordination only happens sparsely in the state space. The agents, for the most part, act independently from the other agents in the environment. Therefore, the authors described Decentralized Sparse Interaction MDPs (Dec-SIMDPs) where the problem is broken down into individual MDPs and Dec-MMDPs. The assumption is that the Dec-MMDPs will be smaller and therefore easier to solve while the remaining individual MDPs will be easily solved.

The research in Dec-SIMDP utilizes a property within team planning in which the individual

team member can often plan independently of the team without explicit coordination. This property helps improve the computational complexity of solving the Dec-MMDP. In a similar perspective, we want to understand the role of the individual robot and proactively check if it can replan locally without the team planner in order to reduce communication, computation, and failing.

9.1.7 Rationale

Rationales have been described as the reasoning behind the choices made by the planner. They can be viewed as goals, decisions, or constraints used by the planner [Polyak and Tate, 1998]. The majority of previous work on rationales has been investigated in single robot domains. The work has focused on validating the rationales of a plan, monitoring the rationales of a plan, and updating the plan when the rationales change [Kambhampati and Kedar, 1991], [Veloso, 1996], [Veloso et al., 1998]. An approach to speed up planning introduced by [Veloso and Carbonell, 1993] uses problem solving cases which store the reasoning and rationale of previous planning problems. The rationale are entirely derived from the planning process. This approach does not consider replanning, adding this information to the plan, or the addition of multiple robots within the plan. Preliminary work has been done in distributed multi-robot domains using rationales. The multi-robot research focused on each robot creating a rationale graph of relevance to other robots, sharing it with other robots, and updating robots when a rationale is violated [Coltin and Veloso, 2013]. The work did not fully explore the implications for replanning, tightly coordinated individuals, or the distribution of different rationales among other robots.

9.2 Execution, Failures, and Replanning

The function of execution is to take a generated plan and physically achieve the actions within the plan. Specifically, execution is the realization of a plan in the physical space of the robot. This realization can include model errors, changed variables, and/or impossible or infeasible actions that make the plan fail. The standard solution is planning again or repairing the existing plan, both considered under the umbrella term replanning. However, early attempts at handling these inconsistencies between planning and execution focused on eliminating the need for replanning.

9.2.1 Policies

In [Schoppers, 1987], Schoppers formulated Universal Plans for controlling robots in dynamic, unpredictable environments. To create a Universal Plan, the goal conditions must be fully described, and from these conditions a Universal Plan is generated using backward-chaining to create a plan from all possible initial conditions of the world. Universal Plans can then be thought of as a decision tree that describes what action should be executed when certain conditions are met by the world. By another definition, they are a MDP generated through planning and it simply creates a *complete* Finite State Machine (FSM) for the environment. Universal Plans are reactive plans but have strong guarantees that the reactive nature is always progressing towards the goal conditions. Another benefit is that Universal Plans are automatically generated compared to hard-coded FSMs. The core problem of classical planning in Schoppers' opinion is that of its choice to over-commitment when planning. Any failures or sabotage leads to replanning which is not an issue

with Universal Plans. However, there have been multiple issues taken with Universal Plans, for example the storage of a plan for any large scale problem would grow exponentially and become infeasible [Ginsberg, 1989].

Research into conditional planning continued as [Nourbakhsh, 1997] demonstrated the theory and practical side of interleaving conditional planning and execution for autonomous robots. His research focused on advanced conditional planning with the assumption that the solution must have a known maximum execution length. Advanced conditional planning creates a conditional plan for every possible event/change in the world using forward-chaining as compared to back-chaining from the goals like Universal Plans. Interleaving is essentially a way of solving only part of the problem before fully planning out the solution, thereby, attempting to reduce the computational cost of generating a conditional plan. There are three ways by which Nourbakhsh accomplishes this with advanced conditional planning: (i) *Information gain* in the way of longitudinal simplifications, i.e., the robot must do this step in all conditional parts of the plan and therefore can execute the action immediately; (ii) *Assumptions* in the way of lateral simplifications, i.e., simplifying the initial state by making assumptions about the environment or robot that does not lead the robot to failures; (iii) Lastly, *Abstractions* which define ways of looking at the planning problem at different levels. The research demonstrates that by interleaving planning and execution together the computation cost of the robot can be greatly reduced while not significantly moving further from optimality. However, this work, to the author's knowledge, has never been implemented on a multi-robot system in part due to the branching computational cost of advanced conditional planning, which is only further exacerbated by multiple robots.

The research into MMDPs and Dec-MMDPS, see Section 9.1.6 in Team Plan Representations, is also focused on generating all possible actions and states of the world. Therefore, the necessity of replanning is removed as all the agents must do is select the appropriate joint action for a given state. However, more research is needed in reducing the complexity of the problem for MMDPs to become feasible for more complex domains. Another limiting factor is the assumption that the team can determine what state they are in together, which it is not always possible to do because of communication limitations. We did see an improvement in Dec-SIMDPs towards this direction but it has yet been shown to work on large scale problems.

9.2.2 Replanning

All replanning algorithms boil down to monitoring the conditions of the plan that make the current action, or team action, achievable. The differences between the approaches depend on the type of conditions they monitor and who detects the failure.

Centralized

The centralized method has a global view of all the robots, and, provided good communication, can monitor the execution of every robot. The general approach is to let the centralized algorithm detect failures and replan accordingly, e.g., STP, [Browning et al., 2005], MASTNs, [Riley, 2005], and CAST [Yen et al., 2001]. However, in the MASTNs case the individual robot can monitor the time constraints and detect those failures. Similarly, in CAST the Petri nets provide conditional constraints that can be monitored by the individual robot but efforts towards replanning were left to future work.

In [Jensen and Veloso, 1998], the authors describe a method for interleaving deliberative and reactive planning together in highly dynamic multi-robot domains. In such domains, like robot soccer, there is a need for immediate action, and reactive planning with FSM is highly effective. Specifically, reactive planning defines a set of hard coded sensor-action rules that are intended to guide the robot to the goal state. However, there is no guarantee that the goal state can even be reached by the robot given the local nature of reactive planning (and they are often hand coded FSMs). This is where deliberative planning helps as it takes the current state, and it generates a set of actions to get to the goal state. However, deliberative planning is expensive, and, in a highly dynamic domain, the robot might not have enough time to create a plan. Therefore, the authors inter-leave the processes together, allowing the robot to be reactive when in a highly demanding situation and deliberative when the robot has more time. It is still an open problem to automatically determine when to switch without the author's hand-coded conditions. It should be noted that replanning in this work is synonymous with switching the type of planning as the conditions of the environment changed.

Mixed Centralized & Decentralized

In general, communication is used in one way or another to coordinate a *decentralized planning* team of robots. In STEAM, [Tambe, 1997], the reactive plans are all the same, implying some level of centralized creation before execution, and the robots declare their intention to execute some plan, which must be agreed upon if multiple robots are involved. Replanning then only occurs when the individual declares its intention to replan to every robot on the team, and they work together to solve the issue. Market-based multi-robot coordination, [Dias et al., 2006], functions as centralized when the team objective is divided into smaller jobs and when the auctioneer must collect all the bids to decide who is given each task. The computation of the bids is the decentralized part of the market-based approach. Replanning can occur when the individual recognizes its inability to complete the task. However, this can create a ripple effect if the information known to the failing robot is not known to the other robots so each robot consequently will generate an incorrect bid thereby continuing until all team members have been exhausted. For other robots to declare re-auction of an individual's task, they must monitor their progress or heartbeats [Gerkey and Mataric, 2002] [Dias et al., 2004]. This leads to some challenges: trading off the duration of a heartbeat, too short too many auctions, too long no progress; trading off opportunistic auctioning, again too many auctions; and how to communicate progress, forcing some type of team plan representation.

There has been working on implementing on-board replanning [McGann et al., 2008]. The authors have built a frame-work for centralized control while allowing for local replanning on-board of the autonomous underwater vehicles (AUVs). However this work has assumed that the planner and the models used by the centralized planner are already on-board the AUVs. The AUVs are also only replanning for temporal disturbances in the plan rather than rationale-driven choices made by the centralized planner.

9.2.3 Generalization of Replanning

Replanning can be considered as all together a different problem from planning as argued by [Talamadupula et al., 2013] in their general theory of intra-agent replanning. There are two approaches to replanning that have been the focus of the research: *replanning*, which has been defined as com-

pletely starting over the planning process, and *plan repair*, which attempts to reuse old parts of the plan for various reasons, e.g., to make replanning easier. Replanning can then be interpreted as a sub-problem of plan repair, as we will discuss, and so the authors focus on defining a general theory for *plan repair*. Although the research literature has focused strongly on the idea of minimizing the changes between the broken plan and the new plan generated after *plan repair*, minimizing changes is only one way of handling the plan repair problem. For example, minimizing the difference between plans can be changed into constraints on what should remain the same after the replanning phase. These constraints can be turned into soft goals and a partial satisfaction planner can be used to plan for the original goals and the soft goals. Thereby, it attempts to satisfy the replanning constraints, but removes those that cannot be achieved as they are soft, and ensures the plan achieves the goal state. Given this definition the restart planning approach is just replanning without any constraints, or soft goals.

9.3 Learning

Learning in multi-robot problems has grown in part to the numerous parameters that can be changed when considering the problem of controlling multiple robots, see review articles [Stone and Veloso, 2000], [Panait and Luke, 2005]. Supervised learning is not often used because it is difficult to label examples in the complex domains involving multiple robots. Reward-based learning is often used as it can be easier to develop for a domain. In general, the problem of learning for multi-agents is vast, so we explore a select few that have attempted to solve the problem of team planning and individual planning within dynamic environments.

9.3.1 Team-Partitioned, Opaque-Transition Reinforcement Learning (TPOT-RL)

TPOT-RL works within dynamic adversarial and cooperative environments that result in the problem of opaque-transitions where the team does not know when it has transitioned into a new state [Stone and Veloso, 1999]. This makes the problem different from the standard POMDP where the assumption is that the transition to a new state is known. Their approach is layered learning where they build up learned behaviors from the hardware level to the team level. The assumption is that a layered approach reduces the complexity of solving the direct mapping from sensors to outputs for a team of robots. They attempt to reduce the state vector by using action dependent variables for each individual (each robot only needs to learn locally), then they compute the reward given that vector, and then select an action, thereby updating the learned values. The approach uses intermediate rewards to help the learning process. Although it can help the learning process, it is hand designed and is not guaranteed to lead to optimal learning.

9.3.2 Multi-layered Neural Networks

Neural networks have gained traction in many domains as a method for approximating complex functions that cannot easily be computed by a human designed algorithm [Schmidhuber, 2015]. Multiple layered networks and advances in training them have increased neural network's usability in real domains, including robotic domains. They can be trained online or offline with data

collected during the robot's execution. In our case, they are very useful for approximating complex functions like predicting the probability of success of an algorithm given continuous variables that represent the robot's current state.

9.4 Robot Soccer Dribbling

Research into accurate dribbling has been previously studied as a way of maintaining control over the ball while navigating. Researchers have used modified potential fields to avoid non-moving obstacles along with constraints on motion to dribble in the RoboCup Middle-Size League [Damas et al., 2002]. Similar to our omnidirectional soccer robots, researchers have analyzed the kinematics and control needed for dribbling a ball along a path [Li et al., 2007]. The only research that models the dynamics of a multi-body environment uses a physics-based robot motion planner [Zickler and Veloso, 2009]. The downfall of this approach is the enormous computational cost of modeling and predicting every robot in the environment. Our work is unique in that it focuses on developing a method of ball-manipulation with opponent awareness while still being computationally feasible in real-time.

Chapter 10

Conclusion and Future Work

In this chapter, we review the contributions of this thesis, then discuss potential avenues for future work, and finally we conclude with a summary of the thesis.

10.1 Contributions

The key contributions in this thesis are:

Rationale-Driven Plan We introduce the rationale-driven plan that includes the rationale for why actions and their parameters are chosen by a centralized planner. It is critical for the individual executing robot to have access to these rationale so that it can successfully replan locally. We introduce multiple algorithms that can be added at critical points within the search process that add the rationale to the plan. The rationale we introduce are the *processing rationale*, i.e., why an action and its parameters are chosen, what goal this action is relevant to, and what other robots are involved in this action, and the *constraint rationale*, i.e., any fact or assumption that restricts the robot or its possible actions and parameters during the planning process.

Intra-Robot Replanning Algorithm We introduce the intra-robot replanning algorithm to handle the issue of locally replanning within a centralized controlled multi-robot team. Provided the rationale-driven plan, the algorithm determines the invalid rationales of the plan. These rationales are then sorted by some predetermined order. Then the applicable replan policies, i.e., the replan policies that can enable the invalid rationale, are stored in a list and when all of them are determined, the list is sorted based on some metric, learned or predetermined. The replan policies are then executed one after another until the rationales become valid again.

Experimental results in the small-size robot soccer domain We present results within a subdomain of the robot soccer domain where two robots attempt to pass the ball to each other while one or more opposing robots, placed in locations disadvantageous for the passing robots, attempt to steal or intercept the ball. Using the intra-robot replanning algorithm and some rationales related to passing the ball, we evaluate the success of our approach of using multiple replan policies to enable the rationale. We show significant improvements in the ability of the robots to successfully pass the ball to each other when they consider the rationale of the plan.

Experimental results in an autonomous underwater vehicle domain We present results in a simulated environment of the ocean. In this domain, the autonomous underwater vehicles (AUVs) need to travel together in order to sample the water efficiently. However, the ocean currents unknown to the centralized planner often lead to navigation errors for the AUVs as they travel underwater. We show that by using our intra-robot replanning algorithm and different replan policies, we can significantly reduce the total cost of completing the team plan for the AUVs.

Experimental results in a grid domain We present results in a targeted planning domain described in the Planning Domain Description Language. We describe the planning algorithm used to generate the rationale for the rationale-driven plan that is provided to the robot in the domain. We show that the robot can significantly reduce the overall cost of executing the plan if it replans locally using our intra-robot replanning algorithm. We also demonstrate that even with zero cost for communicating with the centralized planner, our method is still more advantageous than simply relying on the centralized planner to solve every failure.

Improve intra-robot replanning through learning We present our method for improving the intra-robot replanning algorithm by sorting the replan policies based on a state-based metric for each replan policy. We train neural networks to learn a function that takes the state of the environment and produces a score for a replan policy. Provided each replan policy has its own trained neural network, we are able to sort the replan policies based on their outputs in order to improve the selection of the best replan policy for the current state of the environment. We present results that clearly show an improvement over using a predetermined order for the replan policies.

10.2 Future Work

In this section, we discuss the many avenues for future work within the intra-robot replanning problem that we consider interesting and worth pursuing.

Alternative Rationale The rationale we focus on in this thesis are related to providing the individual with enough information to handle potential failures while executing the plan. We do not however delve into the rationales of why the other actions are not chosen to be in the plan, why actions are pruned from the planning process, why the goals are ordered in the plan the way they are, or in general, the rationales behind why this plan is chosen rather than an alternative plan. We consider these the alternative rationale and we see them as an opportunity for improving the team beyond failures. The alternative rationale can be used to improve the team's performance when opportunities arise that make other actions or goals feasible when the centralized planner assumed they were not. For example, a door may have been assumed to be closed by the centralized planner, which therefore makes the robot take the longer way around to another door that leads into the room. However, when the robot moves past the door, it sees that the door is not closed, making the alternative rationale invalid. The robot then has a choice to replan locally like our method or call the centralized planner to update it with the new information. Both cases may be seen as an improvement over the simple case of ignoring the change and successfully completing the previously provided plan of using another door that is farther away. Although, determining when such a change would be beneficial requires a thorough analysis of the rationale within the plan; similar research can be found in [Sellner and Simmons, 2006, Schermerhorn et al., 2009].

Sorting the rationale In our intra-robot replanning algorithm, we sort the rationale based on some assumed ordering. However, there may be some benefits to an alternative method for sorting the rationale. Similar to the sorting of the replan policies, there may be a method for learning a better sorting that can improve the individual robot's performance. However, such an approach most likely would need to take into account the replan policies that can enable the rationale. This may change the initial flow of the intra-robot replanning algorithm as the replan policies and the rationales may need to be determined together and then sorted together.

Generating the replan policies In this thesis, we focus on domain-specific replan policies to enable the invalid rationales. An alternative may be to generate the replan policies from the planning domain or description. However, this research may cross the line into generating the entire conditional planning tree to handle all potential failures. We discuss such research in our related work in Chapter 9. An effective method would need to concentrate on generating replan policies for failures and rationales in a general way and not be specific to a single part of the planning tree. This research could help with further automating the process of using intra-robot replanning in complex domains.

Reinforcement learning In this thesis, we focus on supervised learning where we collect data from multiple runs using a particular replan policy. However, the robot could attempt to learn a value metric for the replan policy online using reinforcement learning. Going this route would bring up relevant questions of exploration and exploitation when using the multiple replan policies. Balancing those factors, this method would be of value for most likely using less data while still improving team performance.

10.3 Summary

This thesis introduces the rationale-driven plan, several key algorithms for generating rationales, and the intra-robot replanning algorithm. The rationale-driven plan includes the rationales on why the centralized planner chooses the actions and their parameters within the provided plan. The algorithms we introduce for generating the rationales are used at key points within a typical action based planner. The rationale-driven plan is provided to the individual robots that then uses our intra-robot replanning algorithm to solve failures during the execution of the plan. Along with the rationale-driven plan and domain-specific replan policies, the individual robots replan locally and can improve the team performance over the previous method of relying on the centralized planner for fixing failures. We demonstrate our approach in three different domains to highlight the generality of our approach and the benefits across different domains. We further improve our results by learning a value metric for each replan policy using trained neural networks, specifically in the autonomous underwater vehicle domain. Ultimately, we tackle the problem of replanning locally within a centralized controlled team by providing each individual robot with the rationales of the centralized controller, with the ability to replan locally using our intra-robot replanning algorithm, and with a set of domain-specific replan policies.

Bibliography

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

[Bernardini et al., 2017] Bernardini, S., Fox, M., and Long, D. (2017). Combining temporal planning with probabilistic reasoning for autonomous surveillance missions. *Autonomous Robots*, 41(1):181–203.

[Bernstein et al., 2000] Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 32–37, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Biswas et al., 2015] Biswas, J., Cooksey, P., Klee, S., Mendoza, J. P., Wang, R., Zhu, D., and Veloso, M. (2015). Cmdragons 2015 extended team description paper. In *Robocup 2015*.

[Biswas et al., 2014] Biswas, J., Mendoza, J. P., Zhu, D., Choi, B., Klee, S., and Veloso, M. (2014). Opponent-Driven Planning and Execution for Pass, Attack, and Defense in a Multi-Robot Soccer Team. In *Proceedings of AAMAS'14, the Thirteenth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Paris, France.

[Browning et al., 2005] Browning, B., Bruce, J., Bowling, M., and Veloso, M. (2005). STP: Skills, tactics and plays for multi-robot control in adversarial environments. *Journal of Systems and Control Engineering*, 219:33–52. The 2005 Professional Engineering Publishing Award.

[Coltin and Veloso, 2013] Coltin, B. and Veloso, M. (2013). Towards Replanning for Mobile Service Robots with Shared Information. In *Proceedings of the AAMAS'10 Workshop on Autonomous Robots and Multirobot Systems (ARMS)*, St Paul, MN.

[Cooksey et al., 2016] Cooksey, P., Mendoza, J. P., and Veloso, M. (2016). Opponent-aware ball-manipulation skills for an autonomous soccer robot. In *Proceedings of the RoboCup Symposium*, Leipzig, Germany. Springer. Nominated for Best Paper Award.

[Cooksey and Veloso, 2017] Cooksey, P. and Veloso, M. (2017). Intra-robot replanning to enable team plan conditions. In *Proceedings of IROS'17, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada.

[Cooksey and Veloso, 2018] Cooksey, P. and Veloso, M. (2018). A rationale-driven team plan representation for autonomous intra-robot replanning. In *Proceedings of IROS'18, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain.

[Damas et al., 2002] Damas, B. D., Lima, P. U., and Custodio, L. M. (2002). A modified potential fields method for robot navigation applied to dribbling in robotic soccer. In *RoboCup 2002: Robot Soccer World Cup VI*, pages 65–77. Springer.

[de Koning et al., 2017] de Koning, L., Mendoza, J. P., Veloso, M., and van de Molengraft, R. (2017). Skills, tactics and plays for decentralized multi-robot control in adversarial environments. *Submitted to AMAAS*.

[Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95.

[Dias et al., 2004] Dias, M. B., Zinck, M., Zlot, R., and Stentz, A. (2004). Robust multirobot coordination in dynamic environments. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3435–3442. IEEE.

[Dias et al., 2006] Dias, M. B., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270.

[Fabisch, 2017] Fabisch, A. (2017). Openann. `https://github.com/OpenANN/OpenANN`.

[Farinelli et al., 2017] Farinelli, A., Raeissi, M. M., Marchi, N., Brooks, N., and Scerri, P. (2017). Interacting with team oriented plans in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, 31(2):332–361.

[Ferreira et al., 2018] Ferreira, A. S., Costa, M., Py, F., Pinto, J., Silva, M. A., Nimmo-Smith, A., Johansen, T. A., de Sousa, J. B., and Rajan, K. (2018). Advancing multi-vehicle deployments in oceanographic field experiments. *Autonomous Robots*, page 1–20.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Fritz and McIlraith, 2007] Fritz, C. and McIlraith, S. A. (2007). Monitoring plan optimality during execution. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07, pages 144–151. AAAI Press.

[Gerkey and Mataric, 2002] Gerkey, B. P. and Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, 18(5):758–768.

[Ghallab et al., 1998] Ghallab, M., Howe, A., Knoblock, C., Mcdermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL—The Planning Domain Definition Language.

[Gill, 1962] Gill, A. (1962). *Introduction To The Theory Of Finite State Machines*. McGraw-Hill.

[Ginsberg, 1989] Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Mag.*, 10(4):40–44.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302.

[Jensen and Veloso, 1998] Jensen, R. M. and Veloso, M. M. (1998). Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures*. AAAI Press.

[Jordan, 1994] Jordan, M. (1994). *I Can't Accept Not Trying*. San Franciso, CA: HarperSanFrancisco, c1994.

[Kambhampati, 1990] Kambhampati, S. (1990). A theory of plan modification. In *In Proc. AAAI-1990*.

[Kambhampati and Hendler, 1992] Kambhampati, S. and Hendler, J. A. (1992). A validation-structure-based theory of plan modification and reuse. *Artif. Intell.*, 55(2-3):193–258.

[Kambhampati and Kedar, 1991] Kambhampati, S. and Kedar, S. (1991). Explanation-based generalization of partially ordered plans. In *Proceedings of AAAI*.

[Lavalle et al., 2000] Lavalle, S. M., Kuffner, J. J., and Jr. (2000). Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308.

[Levesque et al., 1990] Levesque, H. J., Cohen, P. R., and Nunes, J. H. (1990). On acting together. In *AAAI*, volume 90, pages 94–99.

[Li et al., 2007] Li, X., Wang, M., and Zell, A. (2007). Dribbling control of omnidirectional soccer robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2623–2628. IEEE.

[McDermott, 2000] McDermott, D. (2000). The 1998 ai planning systems competition. *AI Magazine*, 21:35–55.

[McGann et al., 2008] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2008). T-rex: A model-based architecture for auv control. *3rd Workshop on Planning and Plan Execution for Real-World Systems*.

[Melo and Veloso, 2011] Melo, F. S. and Veloso, M. (2011). Decentralized mdps with sparse interactions. *Artif. Intell.*, 175(11):1757–1789.

[Mendoza et al., 2016] Mendoza, J. P., Biswas, J., Zhu, D., Cooksey, P., , Wang, R., Klee, S., and Veloso, M. (2016). Selectively Reactive Coordinatio for a Team of Robot Soccer Champions. In *Proceedings of AAAI'16, the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, AZ.

[Nourbakhsh, 1997] Nourbakhsh, I. R. (1997). *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, Norwell, MA, USA.

[Omerdic and Toal, 2007] Omerdic, E. and Toal, D. (2007). Modelling of waves and ocean currents for real-time simulation of ocean dynamics. In *OCEANS 2007 - Europe*, pages 1–6.

[Panait and Luke, 2005] Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

[Parker, 1993] Parker, L. E. (1993). Designing control laws for cooperative agent teams. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 582–587. IEEE.

[Pecora and Cesta, 2002] Pecora, F. and Cesta, A. (2002). Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*.

[Polyak and Tate, 1998] Polyak, S. T. and Tate, A. (1998). Rationale in planning: causality, dependencies, and decisions. *Knowledge Eng. Review*, 13:247–262.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Riley, 2005] Riley, P. (2005). *Coaching: Learning and Using Environment and Agent Models for Advice*. PhD thesis, Computer Science Dept., Carnegie Mellon University.

[Rubinstein, 2006] Rubinstein, A. (2006). *Lecture Notes in Microeconomic Theory*, pages 87–96. SUNY-Oswego, Department of Economics, Dordrecht.

[Ryan, 2008] Ryan, T. P. (2008). *Modern Engineering Statistics*. Number 124-126. Wiley.

[Schermerhorn et al., 2009] Schermerhorn, P. W., Benton, J., Scheutz, M., Talamadupula, K., and Kambhampati, S. (2009). Finding and exploiting goal opportunities in real-time during plan execution. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 11-15, 2009, St. Louis, MO, USA*, pages 3912–3917.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61(Supplement C):85 – 117.

[Schneider et al., 2014] Schneider, E., Balas, O., Ozgelen, A. T., Sklar, E. I., and Parsons, S. (2014). An empirical evaluation of auction-based task allocation in multi-robot teams. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1443–1444, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Schoppers, 1987] Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'87, pages 1039–1046, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Sellner and Simmons, 2006] Sellner, B. P. and Simmons, R. (2006). Towards proactive replanning for multi-robot teams. In *Proceedings of the 5th International Workshop on Planning and Scheduling in Space 2006*.

[Simmons et al., 2002] Simmons, R., Smith, T., Dias, M. B., Goldberg, D., Hershberger, D., Stentz, A., and Zlot, R. (2002). *A Layered Architecture for Coordination of Mobile Robots*, pages 103–112. Springer Netherlands, Dordrecht.

[Spaan et al., 2002] Spaan, M. T. J., Vlassis, N., and Groen, F. C. A. (2002). High level coordination of agents based on multiagent Markov decision processes with roles. In Saffiotti, A., editor, *IROS'02 Workshop on Cooperative Robotics*, pages 66–73.

[Stone et al., 2006] Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 93–105. Springer.

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In *Proceedings of the third annual conference on Autonomous Agents*, pages 206–212. ACM.

[Stone and Veloso, 2000] Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

[Talamadupula et al., 2013] Talamadupula, K., Smith, D., Cushing, W., and Kambhampati, S. (2013). A theory of intra-agent replanning. *ICAPS 2013 Workshop on Distributed and Multi-Agent Planning (DMAP)*.

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of artificial intelligence research*, 7:83–124.

[Tucker, 2010] Tucker (2010). Ssl dataflow. [Online; accessed 23-October-2017].

[Veloso, 1996] Veloso, M. M. (1996). Towards mixed-initiative rationale-supported planning. In Tate, A., editor, *Advanced Planning Technology*, pages 277–282. AAAI Press.

[Veloso and Carbonell, 1993] Veloso, M. M. and Carbonell, J. G. (1993). Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278.

[Veloso et al., 1998] Veloso, M. M., Pollack, M. E., and Cox, M. T. (1998). Rationale-based monitoring for continuous planning in dynamic environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 171–179, Pittsburgh, PA.

[Yan et al., 2013] Yan, Z., Jouandeau, N., and Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10.

[Yen et al., 2001] Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., and Volz, R. A. (2001). Cast: Collaborative agents for simulating teamwork. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1135–1144. LAWRENCE ERLBAUM ASSOCIATES LTD.

[Zhang et al., 2017] Zhang, S., Jiang, Y., Sharon, G., and Stone, P. (2017). Multirobot symbolic planning under temporal uncertainty. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 501–510, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Zhang et al., 2016] Zhang, Y., Sreedharan, S., and Kambhampati, S. (2016). A formal analysis of required cooperation in multi-agent planning. In *ICAPS*, pages 335–344.

[Zickler et al., 2009] Zickler, S., Laue, T., Birbach, O., Wongphati, M., and Veloso, M. (2009). SSL-Vision: The Shared Vision System for the RoboCup Small Size League. In *Proceedings of the RoboCup Symposium*, pages 425–436, Graz, Austria.

[Zickler and Veloso, 2009] Zickler, S. and Veloso, M. (2009). Efficient Physics-Based Planning: Sampling Search Via Non-Deterministic Tactics and Skills. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 27–33, Budapest, Hungary.