Decentralized Path Planning for Multiple Agents in Complex Environments using Rapidly-exploring Random Trees

by

Vishnu R. Desaraju

B.S.E. Electrical Engineering
University of Michigan (2008)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author ....
Department of Aeronautics and Astronautics
Aug 19, 2010

Certified by....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by ....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Decentralized Path Planning for Multiple Agents in Complex Environments using Rapidly-exploring Random Trees

by

Vishnu R. Desaraju

Submitted to the Department of Aeronautics and Astronautics
on Aug 19, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis presents a novel approach to address the challenge of planning paths for real-world multi-agent systems operating in complex environments. The technique developed, the Decentralized Multi-Agent Rapidly-exploring Random Tree (DMA-RRT) algorithm, is an extension of the CL-RRT algorithm to the multi-agent case, retaining its ability to plan quickly even with complex constraints. Moreover, a merit-based token passing coordination strategy is also presented as a core component of the DMA-RRT algorithm. This coordination strategy makes use of the tree of feasible trajectories grown in the CL-RRT algorithm to dynamically update the order in which agents plan. This reordering is based on a measure of each agent's incentive to replan and allows agents with a greater incentive to plan sooner, thus reducing the global cost and improving the team's overall performance. An extended version of the algorithm, Cooperative DMA-RRT, is also presented to introduce cooperation between agents during the path selection process. The paths generated are proven to satisfy inter-agent constraints, such as collision avoidance, and a set of simulation and experimental results verify the algorithm's performance. A small scale rover is also presented as part of a practical test platform for the DMA-RRT algorithm.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

I would first like to thank Prof. Jonathan How for all of his patience and guidance over the past two years. His ability to identify potential solutions to virtually any problem I encounter has been tremendously helpful, especially as I jumped from one project to the next. All my conversations and meetings with him have helped to shape my research interests. I would also like to thank Prof. Seth Teller for his support in the Agile Robotics project.

I consider myself very fortunate to be a part of the Aerospace Control Lab as well as the Agile Robotics team. The members of the lab and the team are always willing to help, be it a hardware problem, a theoretical problem, or even proofreading this thesis. I offer a special thanks to Dr. Matthew Walter, Brandon Luders, and Georges Aoude for their help in preparing this thesis. I would also like to thank Andrew Whitten, Kenneth Lee, Daniel Levine, Alborz Geramifard, Sameera Ponda, Justin Teo, Sergio Cafarelli, Been Kim, Sachithra Hemachandra, and the rest of my colleagues for their help on various projects and discussions that have led to this thesis. I would also like to thank Kathryn Fischer and Britton Bradley for their administrative assistance, without which we would all be in chaos.

Finally, a sincere thanks to my closest friends, my parents, and my sister for all their support and encouragement. Thanks to them I have been able to maintain some semblance of sanity during even the more trying of times.

This research has been funded in part by the Office of Secretary of Defense under Air Force Contract FA8721-05-C-0002.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Autonomous robotic systems continue to be called upon to perform a multitude of tasks. As with humans, teams of autonomous agents are able to complete multiple tasks in parallel, and moreover, are capable of performing far more complicated tasks than a single agent operating alone. Consider, for example, an automated warehouse as shown in Figure 1-1. The overall objective for the team of autonomous agents operating in this warehouse is to manage the flow of inventory efficiently. To this end, agents are assigned tasks such as taking inventory and moving items. However, as the figure illustrates, agents must also avoid collisions with objects in the environment and other agents while executing these tasks.

This highlights an obvious but fundamental requirement for virtually any multi-agent system: the ability to navigate safely the environment in which the tasks must be performed. In any practical setting, the agents will be limited in their mobility by constrained dynamics, such as a minimum turn radius, as well as actuation limits. These limitations become increasingly important as autonomous teams are deployed in crowded environments, including indoor settings. Consequently, any practical multi-agent path planner must explicitly account for these constraints in order to

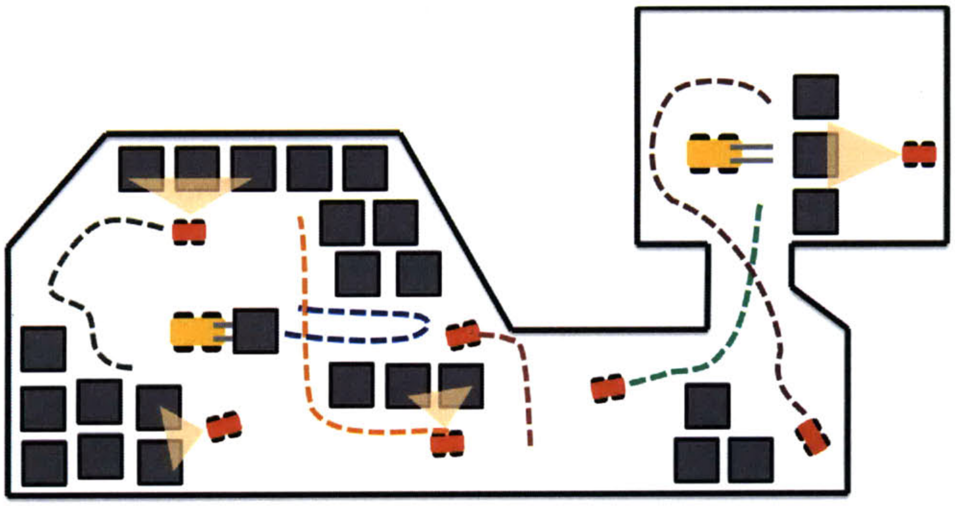


Figure 1-1: A team of autonomous forklifts and ground-guidance robots operating in a constrained warehouse environment

guarantee safe navigation.

As these teams increase in size and operate in larger environments, for example in Intelligence Surveillance, and Reconnaissance (ISR) missions [1], there is a severe increase in the complexity of the planning problem, quickly exceeding the processing capabilities of a single, centralized planner. The communication required for a centralized planner to manage all agents also becomes prohibitive for large scale problems. This motivates the development of decentralized planners, where each agent makes local decisions. However, this approach is not without its own challenges, and any decentralized planner must be careful to avoid conflicting local decisions.

This thesis presents a new multi-agent path planning strategy that is designed with these practical limitations in mind. The underlying path planner is selected to provide an online planning capability that satisfies all constraints, and a decentralized coordination strategy is developed to improve the overall performance of the team while satisfying all safety requirements and maintaining scalability.

1.2 Problem Statement

This thesis examines the problem of planning dynamically feasible paths through complex environments for a team of autonomous agents attempting to reach specified goal locations. The agents are assumed to have constrained, nonlinear dynamics, as seen in any practical system. As a result, agents may be incapable of following paths generated by simple path planners, such as piecewise-linear paths, making dynamic feasibility a crucial requirement. Agents must also be able to navigate safely in cluttered or otherwise complex real-world environments. Furthermore, these paths must satisfy all constraints placed on the interactions between multiple agents, the most common example being collision avoidance between all agents. In general, teams are assumed to consist of many agents that may be operating over a large area, precluding the use of centralized planners to satisfy these requirements. Finally, paths must be selected to minimize a given cost metric, such as travel time or fuel consumption, across all agents in the team.

1.3 Previous Work

1.3.1 Path Planning

There have been numerous path planning algorithms developed for autonomous systems [2], largely due to the fundamental role of path planning in any form of mobile robotics. These approaches are usually categorized as either discrete or continuous path planning algorithms, and methods from both categories have been used with considerable success in different applications. This section provides an overview of several of these popular techniques.

As the name implies, discrete search methods typically begin with a discretized representation of the world, usually in the form of a uniform grid over the environment. Standard search techniques such as dynamic programming [3], A* [4], and variants thereof [5, 6] can then be used to explore this space to find paths. While these approaches often provide completeness and convergence guarantees [7], they typically do not scale well to large problems due to the rapid increase in the dimension of the search space. Furthermore, the dynamic feasible requirement is difficult to enforce, particularly in cluttered or constrained environments, without increasing the resolution of the discretization and thus the dimensionality of the search problem [8].

Several continuous path planning algorithms fall into the category of optimization-based planners. In these algorithms, the path planning problem is cast as an optimal control problem and solved using standard optimization techniques. A notable example of this is the Mixed Integer Linear Program (MILP) formulation in Schouwenaars et al. [9] that, for the multi-vehicle case, easily embeds the collision avoidance constraint in the optimization. Model Predictive Control (MPC) is another optimization-based technique that has been adapted to path planning [10]. MPC uses a model of the system dynamics to predict the system's behavior over a finite horizon and identifies the optimal sequence of control actions for this finite horizon [11], subject to any required constraints [12]. The first action from the sequence is applied and the procedure iterates. While these methods enjoy the benefits of optimization, they also scale poorly with the number of constraints imposed and the complexity of the dynamics and environment, as a result.

A common technique is the use of potential fields to guide agents through the environment. Goal locations contribute attractive forces to the field, while obstacles contribute repulsive forces. Agents then use a feedback control policy to travel along the net force vector [13, 14]. Potential field methods are popular due to their ease of implementation and low computational requirements for many simple scenarios. However, these methods inherently have difficulty with local minima and environments with closely spaced obstacles [15]. Although a potential field without local minima could be used to overcome the primary shortcoming, designing such a field is itself a difficult problem [16].

Sampling-based motion planners constitute another set of continuous planning algorithms that has gained popularity in recent years. These algorithms sample points in the configuration space [17], often via random sampling, and build a set of feasible paths by connecting these points. As a result, the planner is able to quickly explore the configuration space and identify paths without incurring the complexity of constructing the entire space [2]. This improved performance allows these techniques to handle increasingly complex problems, but due to the sampling process, they sacrifice the strong completeness guarantees given by other approaches. Instead, they provide weaker notions such as probabilistic completeness. Probabilistic Roadmap (PRM) planners generate these random samples and connect them using a fast local planner [18], forming a roadmap of feasible paths through the environment. The local planner is then used to connect the agent's start and goal positions to the roadmap before searching the roadmap graph to find a feasible path from start to goal [19].

Another prominent sampling-based path planner is the Rapidly-exploring Random Tree (RRT) algorithm [20]. This planner uses the random sample points to incrementally grow a tree of feasible trajectories from the agent's current configuration. For each new sample point, a branch is grown from the closest node in the tree (as determined by a given distance metric) toward the sample by applying control inputs to a model of the system [21]. These control inputs may be selected in several ways, such as randomly sampling the input space or trying multiple inputs and selecting the one that takes the system model closest to the sample point. After growing the tree for a fixed planning time, the best path is selected according to a given cost metric. Furthermore, this iterative process of growing the tree and updating the path lends itself to online planning.

While the original RRT algorithm is a fast way to plan paths in complex, high-dimensional spaces, it becomes much more useful as an online path planner for autonomous vehicles with a few key changes. Consider, for example, the real-time RRT algorithm proposed by Frazzoli [22] that was later extended by Kuwata et al. in the form of the Closed-loop RRT (CL-RRT) [23–25]. The primary change from the standard RRT is that rather than sampling control inputs to expand the tree, CL-RRT simulates closed-loop system dynamics to select control actions and propagate the states in the tree. As a result, it is able to handle the types of complex and constrained dynamics found in physical systems by embedding the complexity in the simulation. The work presented in this thesis builds upon the CL-RRT algorithm, and as such, a detailed discussion of the algorithm is presented in Chapter 2.

1.3.2 Decentralized Planning

Decentralization is essential for large-scale multi-agent planning problems. Decomposing the multi-agent problem into a set of single agent problems greatly reduces the complexity of each problem. Furthermore, the single-agent path planning techniques described above can then be used to solve these smaller problems. However, although plans generated by individual decision makers may be feasible when considered individually, in order to have global feasibility across all plans there must be some form of coordination between the decision makers [26]. As a result, many coordination strategies have been developed for decentralized path planning, and an overview of several notable methods is presented in this section.

Consensus based approaches allow agents to coordinate actions by exchanging information on quantities of interest that are based on the state of all agents in the team [27]. These quantities of interest may take many different forms, resulting in a variety of consensus based planners. Scerri et al. [28] propose a decentralized space deconfliction strategy for agents in sparse environments. Whenever an agent generates a new path for itself, it sends the path to its teammates to check for conflicts with other known paths. The agent only begins to execute the new path after consensus has been reached on the path's feasibility. While this reduces the amount of local knowledge required to check for path feasibility, it also introduces considerable delay into the planning process. Another notable approach is proposed by Purwin et al. [29, 30] where agents reserve non-intersecting areas of the environment in which they are free to move without the risk of collision. The agents must reach consensus on any modifications to these reserved areas before selecting paths that exit the original area. While this allows for some degree of asynchronous planning (as long as the agents remain within their respective reserved areas), the coordination strategy is conservative; any potential intersection between reserved areas triggers a consensus check, irrespective of whether the intersection would correspond to a conflict in time (and thus a collision) as well.

An alternate approach is reachability based planning [31]. Each agent is provided information on the state of the other agents, for example via vehicle-to-vehicle or vehicle-to-infrastructure communication, and uses this along with models of the other agents to compute their respective reachable sets. In some reachability based methods [32, 33], a set a predetermined rules is then used to select or modify trajectories such that they avoid conflicts in the reachable sets, yielding switching or hybrid control laws. Aoude et al. [34] propose a different approach in which an agent estimates the other agents' reachable sets via CL-RRT simulation. The agent then selects its new path (also via the CL-RRT algorithm) so as to maximize the time to collision with these estimated reachable sets. However, these reachability based methods are typically concerned with agents that are not necessarily cooperative, such as two vehicles approaching an intersection, and thus forced to consider worst-case scenarios in general.

The extension of MPC to multi-agent path planning has also produced some general decentralized planning strategies. Rather than solving one large optimization problem with all agents and constraints embedded, Decentralized Model Predictive Control (DMPC) reformulates the optimization problem as a set of subproblems corresponding to each agent [35]. Then in each MPC iteration, agents solve their respective subproblems to find a new control sequence and communicate the result to the rest of the team. Agents typically plan sequentially (in a fixed order) in each iteration, using the most recently communicated information about the other agents to verify constraints [36, 37]. Venkat et al. provide an analysis of the optimality of these approaches [38]. Citing the restrictive synchronization requirements of the existing approaches, Trodden and Richards propose a different update strategy [39]. With their single subsystem update method, one agent replans in each MPC iteration, while all others continue executing their previously computed plans. This eliminates the synchronization required to cycle through all agents in each iteration. However, there persists the problem of agents taking time to plan, only to keep their current plan or some minor variant of it. Though this typically does not disrupt the team's overall functionality, inefficient use of planning time manifests itself in decreased performance, especially for larger teams.

The primary motivation for this thesis is the current lack of path planning strategies capable of planning efficiently for large teams of agents with the types of complex dynamics and environments found in practical applications.

1.4 Contributions

This thesis presents the Decentralized Multi-Agent Rapidly-exploring Random Tree (DMA-RRT) algorithm as a practical multi-agent path planning strategy. This is an extension of the CL-RRT algorithm [22] to the multi-agent case, retaining its ability to plan quickly even with complex constraints. Moreover, a merit-based token passing coordination strategy is also presented as a core component of the DMA-RRT algorithm. This coordination strategy draws inspiration from the single subsystem update of Trodden and Richards [39], but makes use of the tree of feasible trajectories grown in the CL-RRT algorithm to dynamically update the order in which agents plan. This reordering is based on each agent's incentive to replan (as determined by a potential change in path cost) and allows agents with a greater incentive to plan sooner, thus reducing the global cost and improving the team's overall performance. An extended version of the algorithm, Cooperative DMA-RRT, is also presented to introduce some cooperation between agents during the path selection process. The paths generated are proven to satisfy inter-agent constraints, such as collision avoidance, and a set of simulation and experimental results verify the algorithm's performance.

The structure of this thesis is as follows. Chapter 2 reviews the CL-RRT algorithm and details the DMA-RRT algorithm as well as the merit-based token passing approach. Chapter 3 then extends the algorithm with the emergency stop cooperation strategy. Simulation and experimental results for the original and extended DMA-RRT algorithm are presented in Chapter 4. Chapter 5 introduces a new robotic platform that has been developed as part of a multi-agent team that will serve as a practical testbed for the DMA-RRT algorithm. Finally, concluding remarks and suggestions for future research directions are offered in Chapter 6.

Chapter 2

Decentralized Multi-Agent RRT

As discussed in Chapter 1, the Closed-loop RRT algorithm is very effective at planning paths quickly for a single agent subject to complex constraints. This chapter reviews the CL-RRT algorithm and then extends this planning capability to a team of cooperative agents by embedding the CL-RRT in a framework that manages interactions between multiple agents. As a result, only minimal changes to the CL-RRT algorithm are required for the multi-agent case. Furthermore, a merit-based token passing strategy is presented to improve system performance. This coordination strategy takes advantage of the CL-RRT mechanics and intelligently modifies the order in which agents replan. The resulting Decentralized Multi-Agent RRT (DMA-RRT) algorithm preserves the benefits of planning using CL-RRT while also improving collective performance and guaranteeing that the constraints imposed between agents are satisfied at all times.

2.1 Assumptions

This section outlines the key assumptions made in developing the DMA-RRT algorithm presented below in Section 2.4. The extension to the algorithm presented in the next chapter as well as the ideas discussed in Section 6.2 lay the foundation for relaxing some of these assumptions.

2.1.1 Network

Network connectivity plays a crucial role in the performance of any algorithm for multi-agent systems. It is assumed that all agents involved form a fully connected network. Allowing each agent to talk directly to all other agents allows for fast information transfer with minimal disruption to the algorithm's core functionality. Teams consisting of a few agents operating in a relatively small environment, such as an outdoor storage depot or inside a building, could reasonably form a fully connected network. Of course, this assumption is much less likely to hold for large-scale scenarios where network connectivity is limited by the agents' communication radius, but in these scenarios it is often reasonable to assume that inter-agent constraints only need to be enforced between agents that are sufficiently close, i.e. within the communication radius. The sub-network formed by agents within the communication radius can then be treated as fully connected. However, the problem of planning with multiple sub-networks is beyond the scope of this thesis and is a topic of future research.

Communication on the network is also assumed to be lossless, and any delays introduced are assumed to be negligible. These properties maintain data consistency between agents and can be enforced with the appropriate selection of communication protocols [40].

2.1.2 Agent Models

Each agent is assumed to have a model of its own dynamics to use as in the CL-RRT algorithm. Furthermore, each agent's model is assumed to be known to all other agents, allowing for heterogeneous teams. While accurate models will improve performance, the robustness to modeling error [41] provided by the closed-loop nature of the CL-RRT transfers to the DMA-RRT algorithm, potentially allowing even relatively simple models to work well.

2.1.3 Environment

The environment that the team of agents operates in is assumed to be known and only contain static obstacles. Since the behavior of the agents can be predicted accurately given a model of dynamics and appropriate communication between agents, this assumption enables each agent to anticipate the state of the world for some time into the future and plan accordingly.

2.1.4 Inter-Agent Constraints

Finally, the set of constraints imposed between agents, such as collision avoidance or rendezvous requirements, are assumed to be symmetric between agent pairs. That is, if any agent i is positioned such that it satisfies the constraint with any other agent j, then agent j's constraint with agent i must be satisfied automatically. In the case of collision avoidance, this implies that all agents must maintain the same minimum separation distance and compute this distance in the same way.

2.2 Closed Loop RRT

This section provides an overview of the Closed-loop RRT algorithm [24, 25] described in Section 1.3. The CL-RRT algorithm is considered here due to its capability to quickly plan paths for vehicles with constrained, nonlinear dynamics operating in constrained environments. It is also easily adapted to handle other moving agents [34]. It was successfully demonstrated at the 2007 DARPA Urban Challenge (DUC) as the path planning system for Team MIT's vehicle, Talos [23, 25], underscoring its utility as a practical, online path planner. The CL-RRT algorithm can be broken into three component algorithms: the tree expansion process, the main execution loop, and the lazy check for feasibility.

2.2.1 Tree Expansion

Algorithm 2.1 CL-RRT: Tree Expansion

```
1: Sample point $x_{sample}$ from the environment
2: Identify nearest node $N_{near}$ in tree
3: $k \leftarrow 0$
4: $\hat{x}(t+k) \leftarrow$ last state of $N_{near}$
5: while $\hat{x}(t+k) \in \mathcal{X}_{free}(t+k)$ and $\hat{x}(t+k)$ has not reached $x_{sample}$ do
6:     Compute reference input $\hat{r}(t+k)$ from $x_{sample}$
7:     Compute control input $\hat{u}(t+k)$ from feedback control law (2.1)
8:     Compute next state $\hat{x}(t+k+1)$ from propagation model (2.2)
9:     $k \leftarrow k+1$
10: end while
11: $N \leftarrow \hat{r}_{final}$
12: for each feasible node $N$ produced do
13:     Update cost estimates for $N$
14:     Add $N$ to tree
15: end for
```

The tree expansion process (Algorithm 2.1) for the CL-RRT algorithm begins by randomly sampling a point x_{sample} from the set $\mathcal{X}_{free}$ of all feasible configurations of the agent in the environment. It then identifies the closest node N_{near} in the tree. This notion of closeness is based on various heuristics that can, for example, bias tree growth towards exploring the environment or selecting lower cost paths [22].

Rather than sampling the input u from a set of valid control inputs as in the standard RRT, the CL-RRT algorithm generates a reference r intended to guide $\hat{x}$ toward x_{sample}. This reference is typically of lower dimension than the vehicle model. In the case of a ground vehicle, this could consist of a set of velocity commands and a two-dimensional path for the steering controller to follow. It then uses a feedback control law of the general form

$$u = \kappa(r, x) \tag{2.1}$$

to generate control inputs u that will track the reference r given the agent state x. These inputs are fed into a model of the agent's dynamics

$$\dot{x} = f(x, u) \tag{2.2}$$

to propagate the agent's estimated state $\hat{x}(t+k)$ forward one timestep.

If the propagation model matches the actual system dynamics, selecting $\kappa(r,x)$ to match the control law used on the system will result in trajectories that are not only dynamically feasible, but also match the behavior of the system given the same reference. Selecting $\kappa(r,x)$ to be a stabilizing controller allows the CL-RRT to handle systems with unstable dynamics, such as cars and helicopters. It also reduces the error between the predicted states in the tree and the system's actual state while executing the path [41]. The standard RRT, in comparison, is not able to compensate for modeling errors, disturbances acting on the system, or measurement noise, causing this error to grow over time [42].

This path expansion continues until $\hat{x}$ reaches the sample point or becomes infeasible, and a new node N is created from the final reference point $\hat{r}_{final}$ (the last $\hat{r}(t+k)$ computed). While growing a path, intermediate nodes may also be added at user-defined intervals. These nodes provide additional branching points for future paths, thus improving the algorithm's ability to plan through complex environments. A path cost estimate is then computed for each node produced and the path is added to the tree. Nodes serve as waypoints when executing the path; each path is uniquely characterized by a set of waypoints in addition to the dynamics and reference law used to grow it between those points.

2.2.2 Execution Loop

The execution loop of the CL-RRT, outlined in Algorithm 2.2, uses Algorithm 2.1 to find paths that take the agent to some predefined goal region $\mathcal{X}_{goal}$. It first takes the initial state $x_{initial}$ of the agent and initializes the tree of potential paths. Then, during each iteration of the execution loop, the agent's current state $x(t)$ is propagated by the computation time Δt, and the resulting state estimate $\hat{x}(t+\Delta t)$ is used to compute additional potential paths by the tree expansion process described above. This tree expansion process has a fixed duration Δt, hence the state propagation performed beforehand.

Once the new potential paths are added to the tree, the minimum-cost path p^* in

Algorithm 2.2 CL-RRT: Execution Loop

```
1: $t \leftarrow 0, x(t) \leftarrow x_{initial}$
2: Initialize tree with node at $x_{initial}$
3: while $x(t) \notin \mathcal{X}_{goal}$ do
4:     Update current state $x(t)$
5:     $\hat{x}(t + \Delta t) \leftarrow x(t)$ propagated by $\Delta t$
6:     while elapsed_time $< \Delta t$ do
7:         Grow tree using Algorithm 2.1
8:     end while
9:     if no paths exist in tree then
10:        Apply safety action and goto line 19
11:    end if
12:    Select best path $p^*$ from tree using cost estimates
13:    Recheck feasibility of $p^*$ using Algorithm 2.3
14:    if rechecked $p^*$ is feasible then
15:        Apply $p^*$
16:    else
17:        Goto line 9
18:    end if
19:    $t \leftarrow t + \Delta t$
20: end while
```

the tree is selected. Although the tree expansion process guarantees that all paths added to the tree satisfy the current constraints, these constraints may vary over time in general. To prevent the selection of paths that have become infeasible, the lazy check described in Algorithm 2.3 is performed on p^* before confirming it as the agent's new path. If p^* is infeasible, the process repeats until a feasible path is found or the entire tree is exhausted. If at any time the tree contains no potential paths, a predefined safety action, such as applying maximum braking along the last known feasible path, is executed to keep the agent in $\mathcal{X}_{free}$ [43].

Lazy Check

The lazy check mimics the tree expansion process, but instead of generating new nodes from random samples, it takes the nodes defining p^* and re-simulates the agent's trajectory through them. If at any point the simulated state $\hat{x}(t + k)$ violates a constraint, the path is pruned beyond the last node tested. Thus p^* is guaranteed to be feasible once the lazy check is complete, confirming the property that the CL-RRT

Algorithm 2.3 CL-RRT: Lazy Check

```
1: for each node $N_i \in p^*$ do
2:     while $\hat{x}(t+k) \in \mathcal{X}_{free}(t+k)$ and $\hat{x}(t+k)$ has not reached $N_i$ do
3:         Compute reference input $\hat{r}(t+k)$ from $N_i$
4:         Compute control input $\hat{u}(t+k)$ from feedback control law
5:         Compute next state $\hat{x}(t+k+1)$ from propagation model
6:         $k \leftarrow k+1$
7:         if $\hat{x}(t+k) \notin \mathcal{X}_{free}(t+k)$ then
8:             Remove $N_i$ and all children from $p^*$, goto line 11
9:         end if
10:    end while
11: end for
```

algorithm only returns paths that satisfy all constraints.

2.3 Coordination Strategy

The most straightforward approach to decentralized, multi-agent path planning would be to allow all agents to continuously plan their own paths subject to constraints imposed by the other agents' paths. While this has the benefit of being a fully asynchronous approach, it would could very quickly lead to conflicting plans if one agent modifies the constraints while another is planning.

For example, consider the simple example in Figure 2-1. Agents 1 and 2 begin with feasible plans, but then both of them find alternate plans that appear to be safe given their current knowledge of the other agent. Since this is fully asynchronous, the agents are allowed to update their plans at any time. However, if both update at the same time, the selected paths are no longer safe since the constraints imposed while selecting the paths do not reflect the constraints present when the agents begin execution.

To compensate for this difficulty, this section presents a coordination strategy to ensure that the decisions taken by individual agents do not conflict. Furthermore, this coordination strategy, while not asynchronous, does not require the level of synchronization where, for example, agents are required to take actions at the same time. The underlying logic for this strategy comes from the single subsystem update

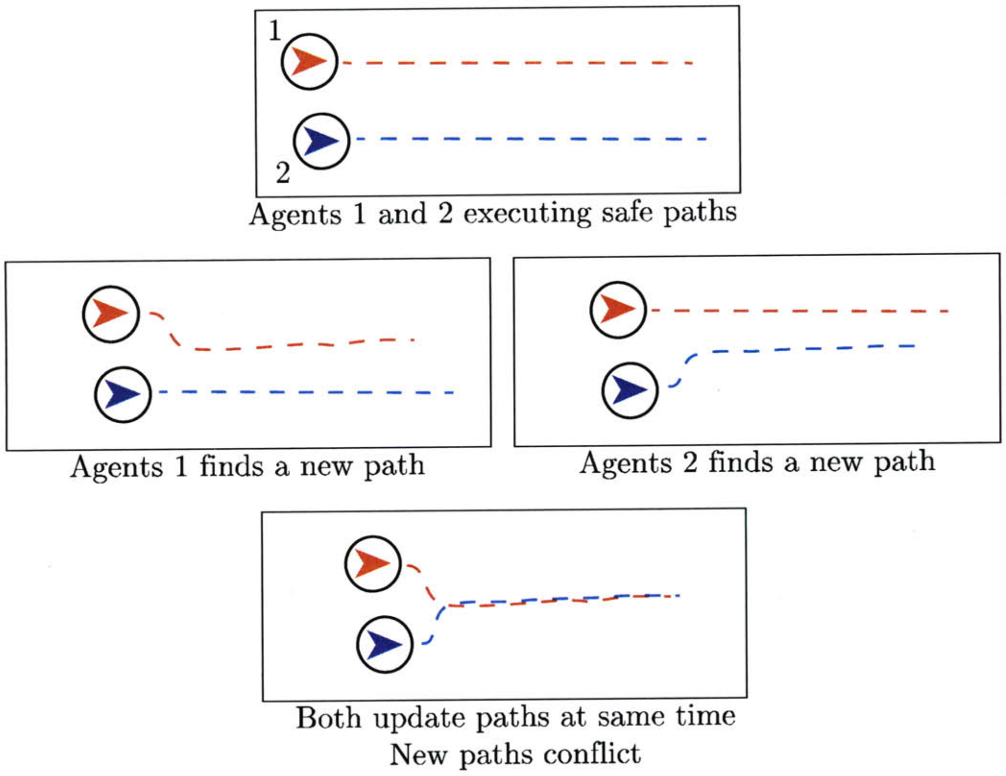

1
2
Agents 1 and 2 executing safe paths
Agents 1 finds a new path
Agents 2 finds a new path
Both update paths at same time
New paths conflict

Figure 2-1: Simple asynchronous planning failure

idea described for DMPC in [39].

As discussed in Section 1.3, the standard DMPC approach is to iterate though the list of agents at each planning iteration and have them update their plans in a fixed sequence. While this could be adapted to use CL-RRT instead of MPC, the synchronization required between agents is impractical for large-scale systems. Agents would be required to wait for the many previous agents in the planning order to finish planning before being allowed to plan, and then must wait for the remainder of the agents to finish before executing the new plan. This greatly slows down the planning process. Instead, the single subsystem update technique as described in Trodden et al [39] iterates through the list, selecting one agent at each planning iteration. This agent is allowed to update its plan, while all other agents are required to keep their previous plans. An immediate consequence of this requirement is that the inter-agent constraints are guaranteed to be fixed while an agent is replanning. This approach reduces the synchronization needed between agents to be able to plan in real time.

2.3.1 Merit-based Token Passing

However, the approach in [39] still requires the agents to cycle through a fixed planning order, regardless of whether an agent will see any substantial benefit from replanning. The alternative approach presented here relies on a measure of Potential Path Improvement (PPI) that reflects an agent's incentive to replan, that is, the improvement in path cost an agent expects to see if allowed to update its plan next.

The single subsystem update strategy is augmented with this PPI information to form a *merit-based token passing* strategy. Rather than iterating through a list of agents, a token is used to identify which agent is allowed to update its plan at each planning iteration. Agents without the token compute their PPI and broadcast these values as bids to be the next token holder. When the current token holder is finished replanning, it passes the token to the agent with the best bid, i.e. the greatest Potential Path Improvement. In the case of a tie, one of the agents in the tie is selected at random to be the next token holder. This effectively produces a dynamic planning order where agents that may benefit the most from replanning are able to do so sooner, without having to cycle through the list and wait for other agents that may have very little incentive to replan. As a result, agents that quickly find paths to reach their goals will typically generate higher bids more often, allowing them to act on these plans sooner and reach more of their goals.

PPI from RRT

Computing the Potential Path Improvement requires the agent to compare costs between the plan it is currently executing and the new plan it would like to switch to. For many path planning algorithms, this would amount to generating a new path just to compute a PPI bid. However, the RRT algorithms provide additional information in the form of the tree of feasible paths, and this can be used to compute the PPI efficiently. In each planning iteration, the agent that is allowed to replan simply continues to run the CL-RRT algorithm and updates its plan accordingly. The other agents continue executing their previously selected plans, but also continue to grow their own tree using CL-RRT. As a result, it is easy for agents to identify new, lower-cost paths in the tree. The difference in path cost between the agent's current path and the best path in the tree is its PPI. For example, if the best path in the tree has a much lower cost than the path the agent is currently taking, the PPI would be large and it would be beneficial to replan soon to select the better path.

2.4 Decentralized Multi-Agent RRT (DMA-RRT)

The DMA-RRT algorithm consists of two components that are run in parallel on each agent. The *individual component* handles the path planning, while the *interaction component* handles all information received from other agents.

2.4.1 Individual Component

The first component, described in Algorithm 2.4, embeds the CL-RRT algorithm. The agent is initialized with some dynamically feasible path, p_0, that satisfies all constraints imposed by the environment and other agents for all time. In the case of a ground vehicle, this path could be as simple as a stopped state, provided it does not conflict with any other agent's trajectory.

One agent is initialized as the token holder, with all others initialized to $Have\_Token \leftarrow false$. As in the CL-RRT algorithm from Section 2.2, the agent grows a tree of feasible trajectories, and at the end of the planning time it identifies the best path, p_k^*, in the tree according to the cost function. The merit-based token passing strategy then determines if the agent will update its plan to p_k^* and pass the token to *winner*, or if it will bid to be the next token holder, as described in the previous section.

If the agent updates its plan, the constraints imposed on the other agents must also be updated accordingly. Due to the closed-loop nature of the RRT, any plan can be characterized by a sparse set of waypoints and closed-loop dynamics (assumed to be known as per Section 2.1.2). Thus, it is sufficient for the agent to broadcast its new waypoints to allow the others to update constraints. This update occurs in the interaction component, described in the next section.

Algorithm 2.4 DMA-RRT: Individual component

```
1: Initialize with $p_0$
2: $Have\_Token \leftarrow false$ except for one randomly selected agent
3: while Agent is active do
4:     Grow CL-RRT (Alg. 2.2), identify best path $p^*_k$ satisfying all constraints (Alg. 2.1)
5:     if $Have\_Token$ then
6:         $p_k \leftarrow p^*_k$
7:         $winner \leftarrow$ agent with best $bid$
8:         Broadcast waypoints of $p_k$ and $winner$ to all agents
9:         $Have\_Token \leftarrow false$
10:    else
11:        $p_k \leftarrow p_{k-1}$
12:        $bid \leftarrow (p_k.cost - p^*_k.cost)$
13:        Broadcast $bid$
14:    end if
15:    $k \leftarrow k + 1$
16: end while
```

2.4.2 Interaction Component

The second component of the DMA-RRT algorithm, described in Algorithm 2.5, controls interaction between agents by handling all incoming information. Communication between agents involves two different message types. The first message contains a list of waypoints corresponding to an updated plan as well as the name of the token *winner*. The current token holder sends this message to all agents once it has finished updating its plan. The second message contains only the PPI value used to bid for the token. This is sent every time an agent other than the current token holder completes the lazy check (2.3) in the CL-RRT and identifies a feasible path p^*.

When a waypoints+*winner* message is received, the agent must update the constraints on its plans. Using a model of the others agent's dynamics, it can simulate the other agent's path along the received waypoints. This greatly reduces the amount of information communicated between agents, as the detailed, time-parameterized path can be reproduced easily from a sparse set of waypoints and the dynamics (Section 2.2.1). This trajectory information can then be used in the CL-RRT's feasibility checks to verify that the path satisfies all constraints imposed on it by the other agents.

Algorithm 2.5 DMA-RRT: Interaction component

```
1: while Agent is active do
2:     Listen for messages
3:     if received waypoints+winner message then
4:         Simulate other agent's trajectory along waypoints, update constraints
5:         if agent is winner then
6:             $Have\_Token \leftarrow true$
7:         end if
8:     end if
9:     if Received bid message then
10:        Update sender's bid
11:    end if
12: end while
```

When the token *winner* receives this message, it can assign itself the token and begin replanning as soon as it updates its constraints. Thus sending the waypoints and the *winner* information together minimizes the delay before the *winner* is allowed to update its path. When a *bid* message is received, it is simply added to the list of bids to check after the next plan update. These bids are stored even when the agent does not have the token so that once it does receive the token, it is not required to wait for bids from the other agents. In addition, the assumption that the network is fully connected, lossless, and introduces negligible delay (2.1.1) allows all agents to receive all messages sent, maintaining data consistency across all agents.

If only a single agent is active, the $Have\_Token$ flag is always *true*, and DMA-RRT reduces to the standard CL-RRT.

2.5 Path Feasibility

As described in Section 2.2, any path added to the tree in the CL-RRT algorithm will be, by construction, dynamically feasible for that agent. In addition, when the best path is selected from the tree, the lazy check ensures that it satisfies all current constraints. Together, these properties guarantee that any path returned by the CL-RRT algorithm will be dynamically feasible and will satisfy all constraints. However, since trajectories returned by the CL-RRT will be of a finite length, an additional constraint is introduced:

For a nominal trajectory $p_k = \{x(k), x(k+1), \dots, x(k+N)\}$ of length $N+1$ timesteps, the agents' states must satisfy

$$\{x(k+M) \in \mathcal{X}_f | \forall M > N\} \tag{2.3}$$

$$\mathcal{X}_f \subseteq \mathcal{X}_{free} \tag{2.4}$$

where the invariant set $\mathcal{X}_f$ can be entirely characterized by a set of time-parameterized waypoints. With this constraint enforced, an agent's behavior is both well-defined and feasible even if it reaches the end of its nominal path, allowing other agents to plan accordingly. Examples of simple invariant terminal sets include a stopped state for a ground vehicle or a loiter pattern for a fixed-wing aircraft, provided the set is feasible when entered [44–47].

These properties allow for a much stronger statement to be proven: any path $p_{i,k}$ generated by agent i at iteration k using the DMA-RRT algorithm is guaranteed to satisfy all constraints at all times.

Theorem 2.1. *Given a set of n cooperative agents and a set of inter-agent constraints satisfying the assumptions of Section 2.1.4, if the initial set of paths $\{p_{i,0}|i = 1, \dots, n\}$ satisfies all constraints, then using the DMA-RRT algorithm, the set of all future paths $\{p_{i,k}|i = 1, \dots, n$ and $k \geq 0\}$ will satisfy all constraints.*

Proof. Assume the set of all agents' paths $\{p_{i,k}|i = 1, \dots, n\}$ at planning iteration k, satisfies all constraints. Then at iteration $k+1$, let agent j be the token holder. Agent j updates its path

$$p_{j,k+1} = p^*_{j,k+1} \tag{2.5}$$

while all other agents keep their existing paths

$$\{p_{i,k+1} = p_{i,k} | i \neq j\} \tag{2.6}$$

Since $p_{j,k+1}$ is generated by the CL-RRT algorithm, it is guaranteed to satisfy all constraints (Section 2.2.2). As a result, the set of constraints imposed on other agents is only changed such that their existing plans continue to satisfy it (Section 2.1.4). Then $p_{i,k}$ satisfying all constraints implies that $p_{i,k+1}$ satisfies all constraints for $i \neq j$. Thus, the set of paths $\{p_{i,k+1} | i = 1, \ldots, n\}$ at iteration $k+1$ satisfies all constraints. Since, by assumption, $\{p_{i,k} | i = 1, \ldots, n\}$ satisfies all constraints for $k = 0$, then by induction, $\{p_{i,k} | i = 1, \ldots, n\}$ satisfies all constraints for all $k \geq 0$ □

2.6 Summary

The Decentralized Multi-Agent RRT (DMA-RRT) algorithm presented in this chapter is a combination of two core components. The underlying path planner is the Closed-loop RRT (CL-RRT), selected for its ability to quickly find dynamically feasible paths even in complex environments with constrained, nonlinear agent dynamics. The second component is a new coordination strategy that uses merit-based token passing to create a dynamic planning order in which agents that may be able to improve their paths significantly are allowed to plan more often. This increases the efficiency of the team and their overall performance. The DMA-RRT algorithm is proven to satisfy all constraints for all time, for appropriate initial conditions. However, no completeness guarantees can be given for the DMA-RRT algorithm as it is, since the individually greedy path selection process can lead to deadlocked situations where agents prevent each other from reaching their goals. An extension to the DMA-RRT algorithm is presented in the next chapter to introduce cooperation between agents. Simulation and experimental results are then presented in Chapter 4.

Chapter 3

Cooperative DMA-RRT

Although the DMA-RRT algorithm as described in the previous chapter implements a coordination strategy using merit-based token passing, each agent is only motivated to minimize its own path cost when selecting a plan. This individually greedy approach does not necessarily minimize the global cost, i.e. the total cost across all agents in the team. This chapter introduces a cooperation strategy that allows an agent to modify its own path as well as the path of another agent in order to minimize the combined path cost. Truly minimizing the total cost would typically require modifying plans for all agents, as in the case of a centralized planner. However, this pairwise coordination begins to mimic the advantage of the centralized planner without sacrificing the decentralized implementation or incurring the communication penalty. This Cooperative DMA-RRT algorithm introduces *emergency stop* nodes along each agent's path where the agent could safely terminate the path if asked to by another agent. The decision to terminate another agent's path is based on a cost comparison. In addition, agents are assumed to be rational and cooperative, such that agents will stop if asked to do so by another agent.

3.1 Modified DMA-RRT Algorithm

While the emergency stop based cooperation strategy affects the interaction between agents, most of the algorithmic changes required to implement the emergency stop

Algorithm 3.1 DMA-RRT: Individual component with emergency stops

```
1: Initialize with $p_0$
2: $Have\_Token \leftarrow false$ except for one predetermined agent
3: while Agent is active do
4:     Grow CL-RRT ignoring other agents' paths, identify best path $p_k^*$
5:     if $Have\_Token$ then
6:         if $p_k^*$ conflicts with some $agent\_j$ then
7:             Check emergency stops (Algorithm 3.2)
8:         end if
9:         if $p_k^*$ conflicts with some other $agent\_j'$ then
10:            $p_k^*$ pruned at conflicting node to avoid conflict with $agent\_j'$
11:        end if
12:        Identify emergency stop nodes on $p_k^*$
13:        $p_k \leftarrow p_k^*$
14:        if $agent\_j$'s plan was modified then
15:            $next\_agent \leftarrow agent\_j$
16:        else
17:            $next\_agent \leftarrow$ agent with best $bid$
18:        end if
19:        $winner \leftarrow$ agent with best $bid$
20:        Broadcast waypoints of $p_k$ (with emergency stops) and $winner$ to all
           agents
21:        $Have\_Token \leftarrow false$
22:    else
23:        $p_k \leftarrow p_{k-1}$
24:        $bid \leftarrow (p_k.cost - p_k^*.cost)$
25:        Broadcast $bid$
26:    end if
27:    $k \leftarrow k + 1$
28: end while
```

logic occur in the individual component of the DMA-RRT algorithm (Algorithm 2.4). The modified version of the individual component is presented in Algorithm 3.1, with the additions (highlighted in red) discussed below.

When some *agent_i* selects a plan p_k^*, it can also identify several nodes in the plan where it could stop if necessary. These emergency stop nodes can be selected by taking every n^{th} safe node or selecting the next safe node every n seconds along the path, where safe indicates the agent can stop at the node for all future time without violating any constraints. When the agent broadcasts its plan, it also indicates which nodes in the plan correspond to these emergency stop locations. As with the terminal node on the path, these nodes block off areas where the agent can stop safely for all time. Thus no other agents can select paths going through those areas, unless they do so before *agent_i* is expected to arrive. These nodes already have costs associated with them from the tree-growing process of the CL-RRT, so other agents know the cost of forcing this agent to stop at one of these nodes.

Paths are no longer tested for conflicts with other agents' paths when growing the tree. Instead, the algorithm relies on the Emergency Stop Check described in Algorithm 3.2 to maintain feasibility relative to the other agents paths. If the path selected by *agent_i* does not come into conflict with another agent's path, the algorithm proceeds normally. However, if the path conflicts with some *agent_j*, the the Emergency Stop Check is performed.

There are several options if the path conflicts with more than one agent, ranging from ignoring emergency stops for all agents to checking all combinations of stops for all agents involved. The option of ignoring all emergency stops defeats the purpose of introducing this cooperation strategy, particularly in scenarios where the agents are operating in close proximity to each other, while with the latter option, the number of combinations that must be considered grows exponentially with the number of conflicting agents. The approach considered here is an intermediate option where emergency stops are checked for the first conflicting agent, *agent_j*, but not for any subsequent conflicting *agent_j'*.

For each of the emergency stop nodes *agent_j* published, *agent_i* will identify the last node in its path p_k^* that it can stop at. When performing this check, the only the emergency stops considered viable are those that are far enough ahead in time that *agent_j* will not have passed them by the time *agent_i* is done planning.

Since *agent_i* knows the costs associated with each of its own nodes, as well as *agent_j*'s costs for stopping early, it can select the combination of terminal node and emergency stop node that yields the best total cost across both agents. If this requires *agent_j* to stop early, a message is sent indicating at which emergency stop node it should terminate its path.

Then *agent_i* passes the token to *agent_j* so that it can update its plan but adds

Algorithm 3.2 Emergency Stop Check

```
1: if $check\_estops$ then
2:     for all viable emergency stop nodes $N_l$ in $agent\_j$'s path do
3:         Assuming $agent\_j$ stops at $N_l$, find last node in $p_k^*$ that is safe to stop at
4:         $TotalCost_l$ = cost of path ending at this node + $agent\_j$'s cost to stop
           at $N_l$
5:     end for
6:     Select the terminal node and $N_l$ that minimize $TotalCost_l$
7:     if $N_l$ is not $agent\_j$'s original terminal node then
8:         Send message to $agent\_j$ to stop at $N_l$
9:     end if
10:    if selected terminal node is not the original terminal node then
11:        $p_k^*$ pruned past new terminal node
12:    end if
13: else
14:    $p_k^* \leftarrow p_k^*$ pruned to satisfy all constraints
15: end if
```

the caveat that *agent_j* cannot use the emergency stop logic on this planning iteration. This prevents cycles between *agent_i* and *agent_j* that may be triggered by each agent finding an incrementally better path that forces the other agent to stop early. At the very least, *agent_j* will then broadcast the truncated version of its old plan, thus updating all other agents and preventing them from detecting conflicts with the path segment that was removed.

Only minimal changes are required to Algorithm 2.5, the interaction component of the DMA-RRT, to implement this emergency stop logic. These changes are highlighted in red in Algorithm 3.3. When an agent receives an *estop* message indicating it must terminate its current path early, it prunes all nodes in its current path p_k after the emergency stop node specified in the message. It also sets the *check_estops* flag to *false* so that it will not attempt to check against other agents emergency stop nodes during its next planning iteration, as described above.

The resulting algorithm, DMA-RRT with emergency stops, is illustrated by the following example.

Algorithm 3.3 DMA-RRT: Interaction component with emergency stops

```
1: while Agent is active do
2:     Listen for messages
3:     if received waypoints+$winner$ message then
4:         Simulate other agent's trajectory along waypoints, update constraints
5:         if agent is $winner$ then
6:             $Have\_Token \leftarrow true$
7:         end if
8:     end if
9:     if Received $bid$ message then
10:        Update sender's $bid$
11:    end if
12:    if Received $estop$ message then
13:        Terminate $p_k$ at node stop specified in $estop$
14:        $check\_estops \leftarrow false$
15:    end if
16: end while
```

3.2 Example

Figure 3-1 shows a simple two-agent scenario where this cooperation strategy would be beneficial. The agent in green is the current token holder. The first column shows the performance of the original DMA-RRT algorithm. Agent 2 plans a path that ends at the entrance to the narrow hallway. Agent 1 then finds a path to the goal but is forced to terminate the path in the hallway to avoid a conflict with Agent 2's path (assume in this scenario that Agent 2 will arrive first due to its shorter path). This leads to a deadlock scenario where neither agent is able to progress toward its goal, due to selecting paths purely based on local cost functions.

The second column shows the same setup with the the algorithm modified to use this cooperation strategy. Agent 2 generates the same plan but also identifies two emergency stop nodes, indicated by the gray dots. Then Agent 1 finds the path to the goal and compares the three different scenarios shown in the third panel. The first leaves Agent 2's plan untouched, the second forces Agent 2 to stop at the second emergency stop node, and the third forces it to take its first stop. Since the second option allows Agent 1 to reach its goal without cutting too much off Agent 2's path, that is the option it selects. Following through on these new plans allows Agent 2 to plan a path to the goal as soon as Agent 1 passes.

3.3 Path Feasibility

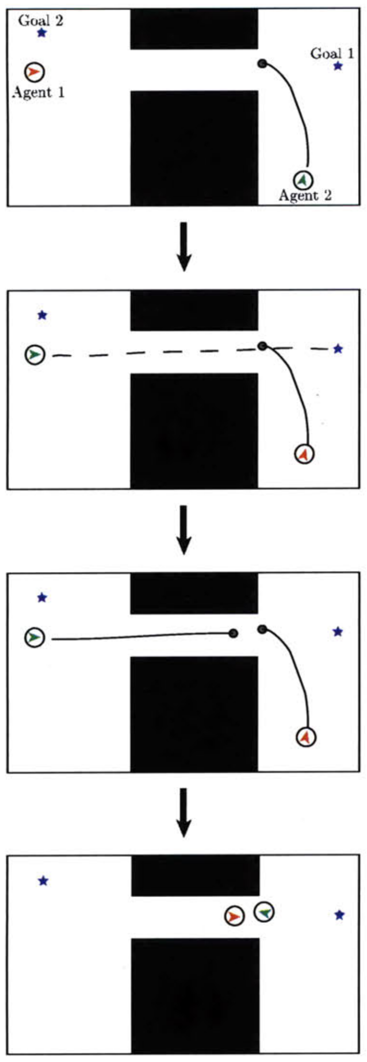
The addition of this coordination strategy to DMA-RRT algorithm preserves the path feasibility guarantee from Theorem 2.1. By construction, all emergency stop nodes satisfy the safe stopping requirement imposed on terminal nodes in the original algorithm. Thus, the modified path ending at an emergency stop node is a path satisfying the terminal node requirement. As with the terminal nodes, these emergency stop nodes are known to all agents, ensuring no other plans will be in conflict with them.

Furthermore, the only emergency stop nodes that are considered are ones that are sufficiently far ahead in time. This buffer compensates for any delays that may arise before the other agent truncates its path, ensuring feasibility even during message passing. Therefore, the modified set of plans across all agents is feasible before, during, and after each planning iteration and the proof of Theorem 2.1 holds.

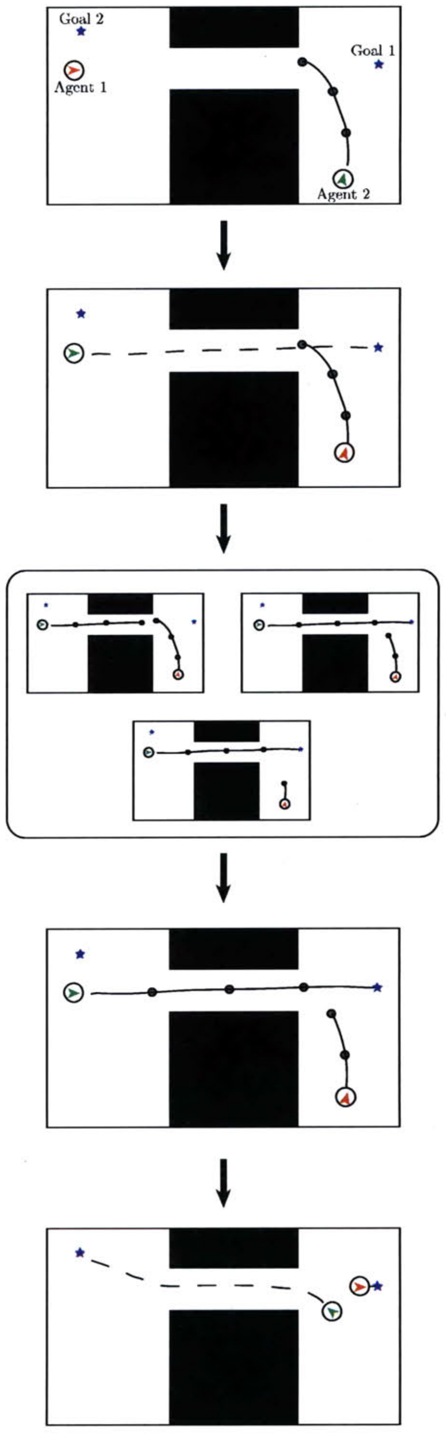
3.4 Limiting Case

As described in Section 3.1, emergency stop nodes for a given path are separated by approximately equal intervals, either in time or by number of nodes. Maximizing the interval size restores the functionality of the original DMA-RRT algorithm. Smaller intervals provide greater flexibility in matching terminal nodes and emergency stops at the cost of evaluating the additional combinations and blocking off more of the environment. In the limiting case where the interval size is taken to be zero, all nodes in the path can be treated as emergency stops. Although this will greatly increase the computational requirements to check all the emergency stops, it also allows agents to stop almost anywhere on their current paths without the risk of violating constraints.

A similar approach is described by Purwin et al. [29] where agents reserve non-intersecting areas in the environment where they are free to plan without the risk of



(a) Agent 2 plans a path that prevents Agent 1 from reaching its goal. The two agents reach the ends of their paths, only to be obstructed by the other agent.



(b) Agent 2 plans a path with emergency stops. Agent 1 finds a path to the goal, considers Agent 2's emergency stop nodes and selects the best one. Agent 2 truncates its path and waits for Agent 1 to pass

Figure 3-1: An example scenario with (b) and without (a) the emergency stop logic

colliding with other agents. Whenever an agent needs to travel outside its reserved area, it computes a new desired reserved area and identifies which other agents' areas it would conflict with. The agents then use a priority assignment scheme to determine whether each conflicting agent, including the one instigating the conflict, is allowed to keep its reserved area or if it must be reduced to resolve the conflict. This allows higher priority agents to override the reserved areas of lower priority agents. However, the emergency stop logic retains one distinct advantage over this approach: the emergency stop nodes are time parameterized. If an agent is expected to arrive at a node at time t_1, the section of the environment corresponding to that node is only infeasible for other agents starting at time t_1, rather than blocking the space for all time. This allows agents to plan conflict-free paths through that space for any time $t < t_1$. Other agents planning through that space for $t \geq t_1$ must perform the emergency stop check instead.

Allowing the agent to stop virtually anywhere on its path also introduces an option for relaxing the static environment assumption from Section 2.1.3. If the agents are operating in a world with dynamic objects whose behavior is uncertain but non-adversarial, the ability to stop at any node typically is enough to avoid collisions with these objects.

3.5 Summary

Motivated by the individually greedy approach each agent uses in the DMA-RRT algorithm of the previous chapter, this chapter presents an extension to the algorithm that adds pairwise cooperation between agents. This cooperation is introduced in the form of emergency stop nodes that are placed along the agents paths. These nodes indicate places where the agent can safely stop early if needed. By sharing this information, along with the cost incurred by stopping early, agents are able to select paths that reduce the total cost for a pair of potentially conflicting agents.

Chapter 4

Results

The DMA-RRT algorithm and the emergency stops extension developed in the proceeding chapters aim to improve the overall performance of multi-agent teams. This chapter presents a set of simulation and experimental results from several scenarios involving teams of different sizes and environments of varying complexity in order to verify the performance improvement expected from these algorithms.

4.1 Simulation

4.1.1 Setup

The DMA-RRT algorithm is implemented in real-time Java as an extension of the RRT-Sim software developed at the MIT Aerospace Controls Laboratory (ACL). As the name suggests, this software package provides a modular simulation environment for single-agent path planning using CL-RRTs. Additional details on the RRT-Sim software are available in [48].

To maintain the decentralized nature of the DMA-RRT algorithm, each agent is simulated on a different computer, with all the computers connected to the same local area network. Each agent runs an implementation of Algorithm 2.4 to handle the actual path planning and an implementation of Algorithm 2.5 to share data with other agents. The CL-RRT from RRT-Sim forms the core of the individual component, while network communication capabilities added to RRT-Sim enable the inter-agent communication required in the interaction component.

Inter-Agent Communication

The User Datagram Protocol (UDP) is used for all communication between simulated agents. UDP provides very low latency communication as well as the ability to broadcast and multicast messages to an entire network. This capability facilitates the implementation of a fully-connected network, as stated in the assumptions (Section 2.1.1). The tradeoff with UDP is that it does not in inherently guarantee data reliability, and issues such as packet loss, must be taken into account when selecting a communication protocol. However, UDP communication has been demonstrated, in practice, to work reliably with low volumes of data transfer. The DMA-RRT is designed to broadcast minimal amounts of data (Section 2.4.2), and as such, UDP is a viable option for problems of the scale considered in this section.

The data communicated between agents consists of two types of messages. The first is the bid message, which only contains the name of the sending agent and its PPI bid. This small message is typically 16 bytes long and is sent at approximately 1 Hz (a function of the CL-RRT plan time). The other message is the "waypoints+winner" message described in Section 2.4. This message contains the sender's name, its current state and reference, the nodes defining its current path, and the name of the next token recipient. While the size of this packet varies with the number of nodes in the path, it is typically between 150 bytes and 1500 bytes in length. However, this message is sent much less frequently than the bid message, with the average frequency dependent on the number of agents.

Implementation Details

All agents in the simulation are assumed to be identical two-wheeled, skid-steer vehicles, whose dynamics in the global frame are given by the modified bicycle model

$$\begin{aligned} \dot{x} &= v \cos\theta \\ \dot{y} &= v \sin\theta \\ \dot{\theta} &= \frac{\Delta v}{L_w} \end{aligned} \tag{4.1}$$

where $v = \frac{1}{2}(v_L + v_R)$, the average of the left and right wheel velocities, $\Delta v = v_L - v_R$, the difference between the left and right wheel velocities, and L_w is the distance between the left and right wheels [49]. Since v_L and v_R are also the two control inputs to the agent, no additional velocity controller is needed. The pure-pursuit steering control law is used to track the reference path with a smooth trajectory [23]. This control law, adapted to skid-steer dynamics is given by

$$\Delta v = -v \left(\frac{L_w \sin\eta}{\frac{L_1}{2} + l_a \cos\eta} \right) \tag{4.2}$$

where L_1 is the pure-pursuit look-ahead distance, l_a is the distance of the anchor point from the axle, and η is the angle from the vehicle to the look-ahead point on the reference path. Details on the pure-pursuit controller and its adaptation to skid-steer dynamics are presented in Appendix A.

The inter-agent constraint, collision avoidance, is implemented using RRT-Sim's built-in capability to handle time-parameterized obstacle data, known as Layer Cake. Whenever another agent's path information is received, the corresponding trajectory is computed by simulating the agent's dynamics along the nodes in the path. This trajectory is, by definition, the set of predicted time-parameterized states of the other agent and thus can be embedded in the Layer Cake as a moving obstacle to avoid. Then, without any modifications, the CL-RRT in RRT-Sim is able to check the collision avoidance constraint for all other agents' trajectories when selecting a path. Furthermore, each agent is assigned its own layer in the layercake, allowing the emergency stop logic to identify specific conflicting agents.

Three simulation scenarios are presented in the following sections using this implementation of the DMA-RRT. The results for each are based on 120 minutes of simulation data (12 ten-minute trials), and performance is measured in terms of the average number of goals each agent is able to reach in ten minutes.

4.1.2 Scenario: Ten Agents on Open Map

The first scenario involves a team of ten agents operating in an open environment. Though there are no static obstacles (other than the perimeter), the density of agents in this limited area increases the complexity of the environment through which they must plan. Each agent is assigned one of the initial positions and one of the goal locations in Figure 4-1. One arbitrarily selected agent starts with the token. Upon reaching a goal, agents select the next goal in the list (looping back to the first goal after the tenth) and resume planning.

DMA-RRT without Merit-Based Token Passing

This scenario is first run *without* merit-based token passing. Instead, the agents use a round-robin approach, cycling though a fixed planning order.

Figure 4-2 shows the state of the world as seen by agent 1 (blue circle with red arrow) at one instance in time. The other agents are represented by yellow circles with blue arrows, and the current token-holder's arrow is highlighted[1]. The magenta dots are the waypoint nodes along agent 1's current path. The time-parameterized obstacles produced by the other agents' trajectories are shown by the red-to-green paths. Time is represented by the color gradient from bright red (current obstacles, i.e. the other agents' current positions) to bright green (obstacles at least 10 seconds ahead in time). The gray areas are the safe terminal nodes for each path. These are all based on agent 1's re-simulation of the other agents' trajectories.

On average, each agent reached a total of 12.5 goals in a span of ten minutes. In

[1]The arrow is cyan in this case since one of the other agents has the token. If agent 1 were the current token holder, its arrow would be green instead.

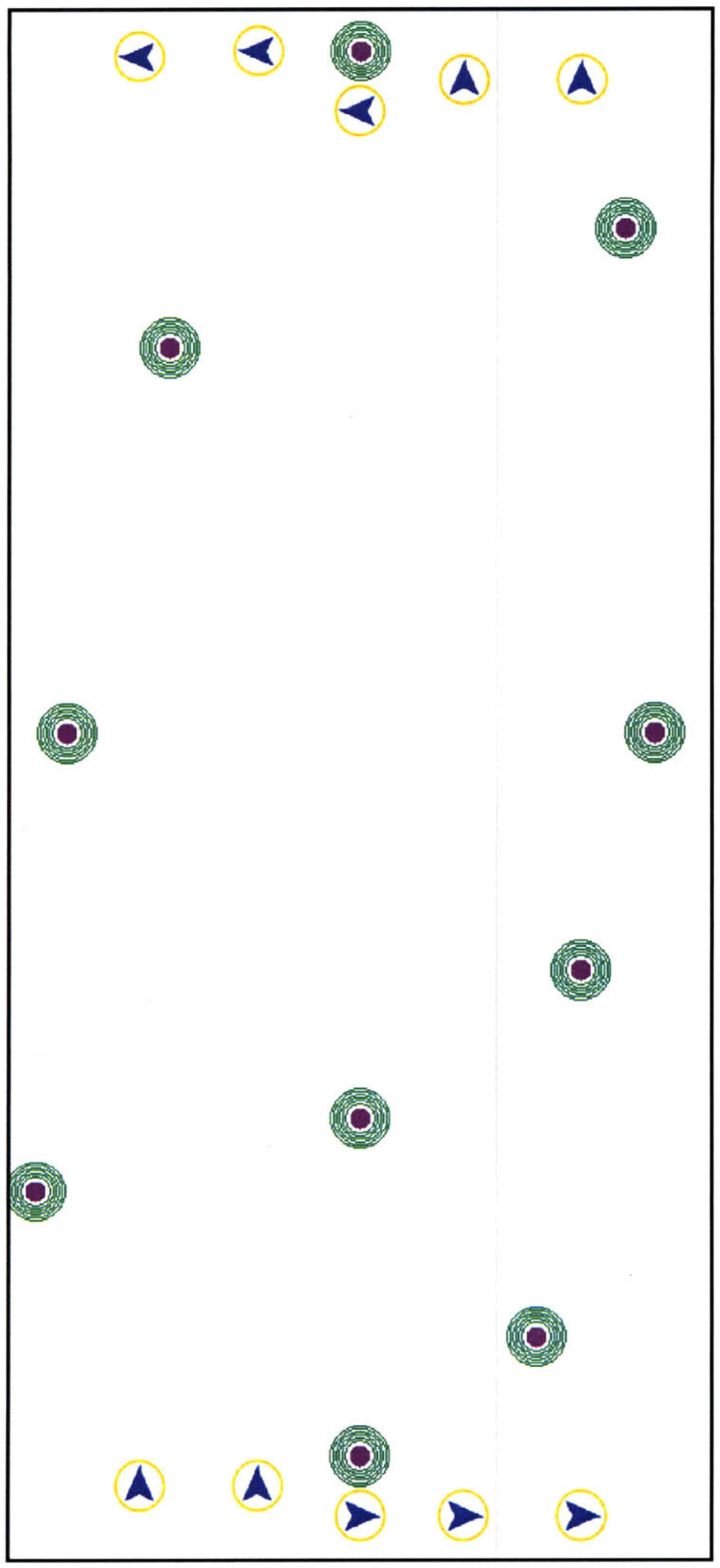


Figure 4-1: Open map with ten initial positions (yellow circles with blue arrows) and ten goal locations (bullseye shapes)

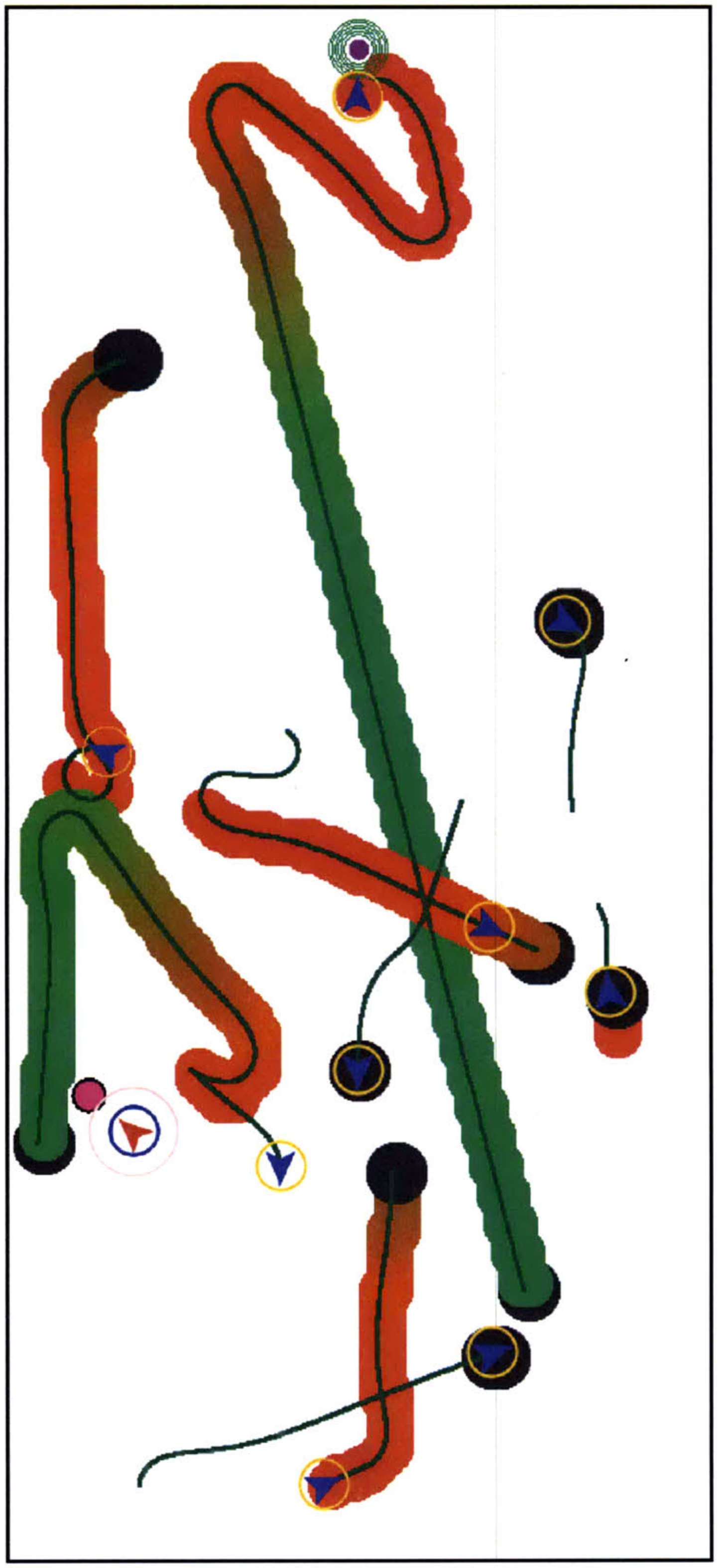


Figure 4-2: State of the world as seen by agent 1 for the open map scenario, with all time-parameterized obstacles displayed

comparison, a single agent operating on the same map averages around 40 goals in ten minutes, so the performance drop per agent due to increasing the number of agents to ten is not even a factor of four, let alone a factor of ten. The fixed planning order is evident in the example replan start times shown in Figure 4-3.

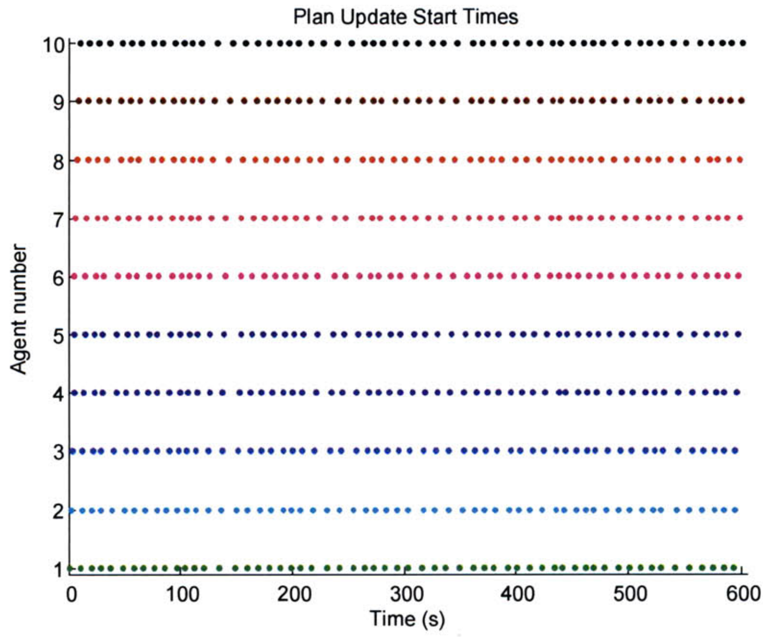
DMA-RRT

The same test is then repeated with the full DMA-RRT algorithm, i.e. using merit-based token passing. Each agent computes its path cost, and thus its bid, based on the path's length in seconds (i.e. travel time). Adding the intelligent coordination policy yields significantly better results. Each agent reached an average of 15.1 goals in ten minutes. The dynamic planning order this token passing strategy produces is also apparent from a comparison of the replan start times, an example of which is shown in Figure 4-4.

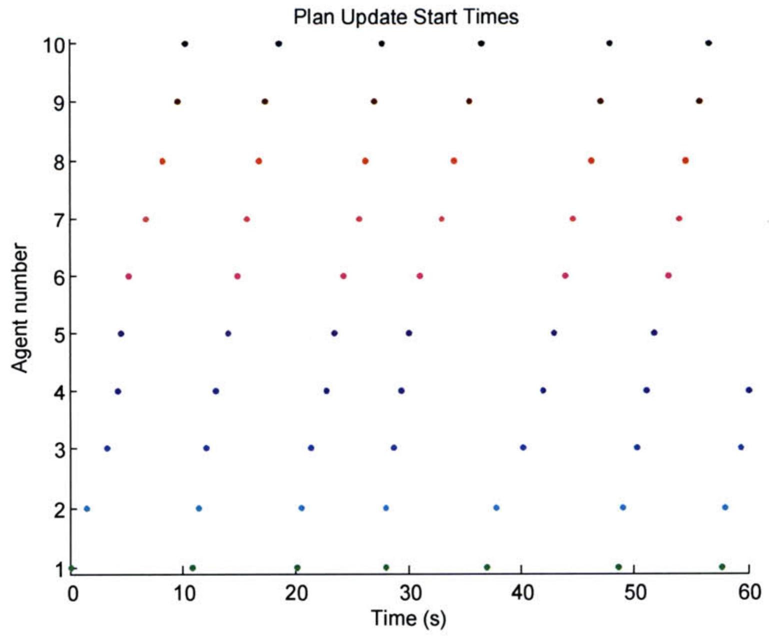
Analysis

This simulation scenario demonstrates the DMA-RRT algorithm's ability to handle large teams of agents. When using the round-robin strategy, all agents are given equal opportunities to plan, irrespective of whether they need to replan at that time. However, if an agent finds a new path after passing the token, it is forced to wait for the token to pass through the entire team before receiving it again. As Figure 4-3 shows, the wait time is typically around ten seconds for this ten-agent scenario. In a more complex scenario with additional agents and obstacles, this delay would only increase.

In comparison, merit-based token passing enables agents to regain the token within a second or two of passing it, if needed. An example of this is in Figure 4-4 around 12 seconds where agent 7 passes the token to agent 3 but wins the bid to get it back from agent 3 immediately after. This reduces the effective "penalty" incurred in the round-robin approach when agents with a large incentive to replan may be blocked form doing so by agents with little incentive. Thus, agents that are able to update to shorter paths can do so much more quickly, reducing the travel time between goals

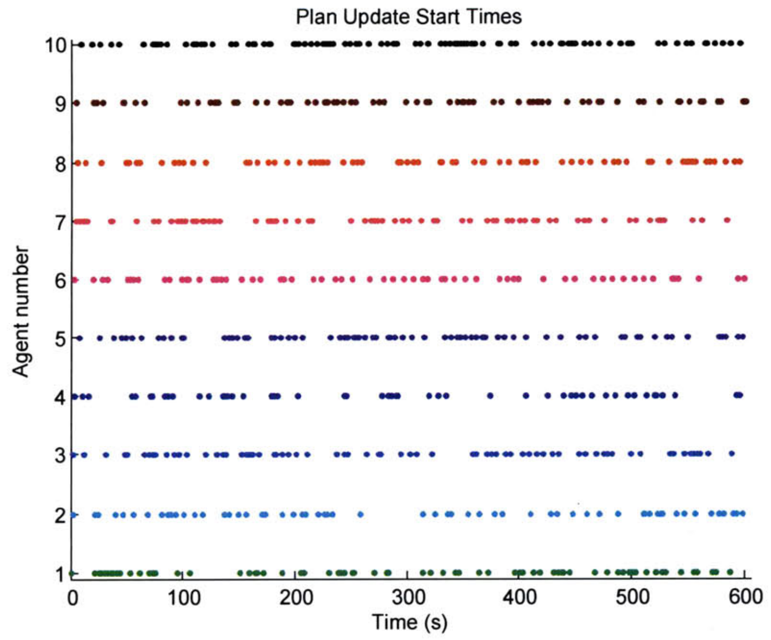


(a) Times at which each agent was allowed to update its plan during one ten-minute simulation

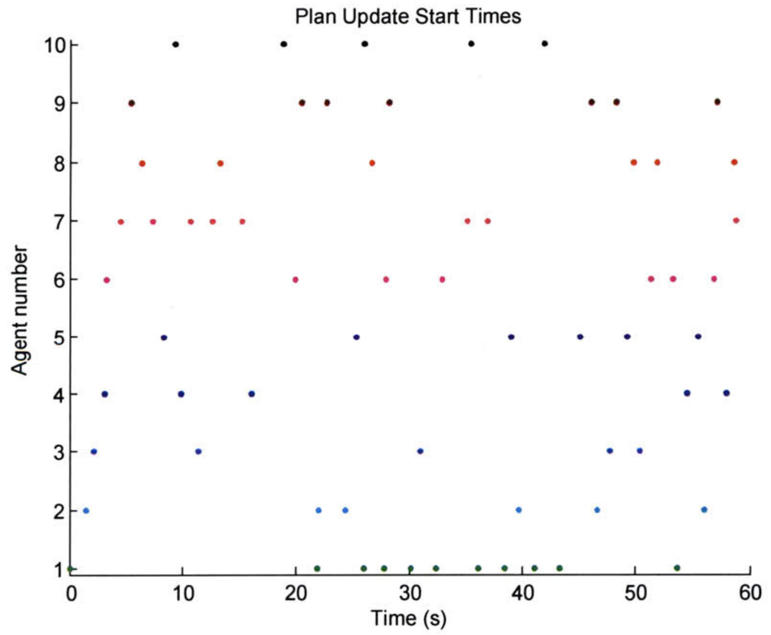


(b) Close-up showing sequential planning order (consistently increasing plan times between each agent in order)

Figure 4-3: Replan start times for round-robin approach (Ten Agents on Open Map)



(a) Times at which each agent received the token to update its plan (i.e. replan) during one ten-minute simulation



(b) Close-up showing dynamic planning order (agents plan more often or less often at different times)

Figure 4-4: Replan start times for merit-based token passing (Ten Agents on Open Map)

and increasing the number of goals that can be reached in a fixed amount of time. The simulation results reflect this quite clearly, with a 20% increase in the average number of goals per agent, from 12.5 to 15.1.

4.1.3 Scenario: Four Agents with Obstacles

The second scenario is in a complex environment containing several static obstacles and two goal locations, as shown in Figure 4-5. Four agents are initialized at the positions shown in the same figure. Each agent is given one of the two goals as its initial target. Upon reaching a goal, the agent selects the goal at the opposite end of the environment and continues planning.

DMA-RRT without Merit-Based Token Passing

Again, this scenario is first run using the round-robin approach. Figure 4-6 shows a snapshot of agent 1's knowledge of the world during a sample run. On average, each agent reached a total of 11.9 goals in a span of ten minutes using the round-robin approach. The fixed planning order is evident in the example start times shown in Figure 4-7.

DMA-RRT

The same scenario is again re-run with the full DMA-RRT. In this case, even with merit-based token passing, each agent averages 11.9 goals in ten minutes. Figure 4-8 shows the times at which the token was received in a sample run.

Cooperative DMA-RRT

Finally, the emergency stop cooperation strategy is enabled and the same scenario is run again. The stop nodes are placed approximately every four seconds along each trajectory, as seen in Figure 4-9. With emergency stops enabled, each agent averaged 13.0 goals in ten minutes.

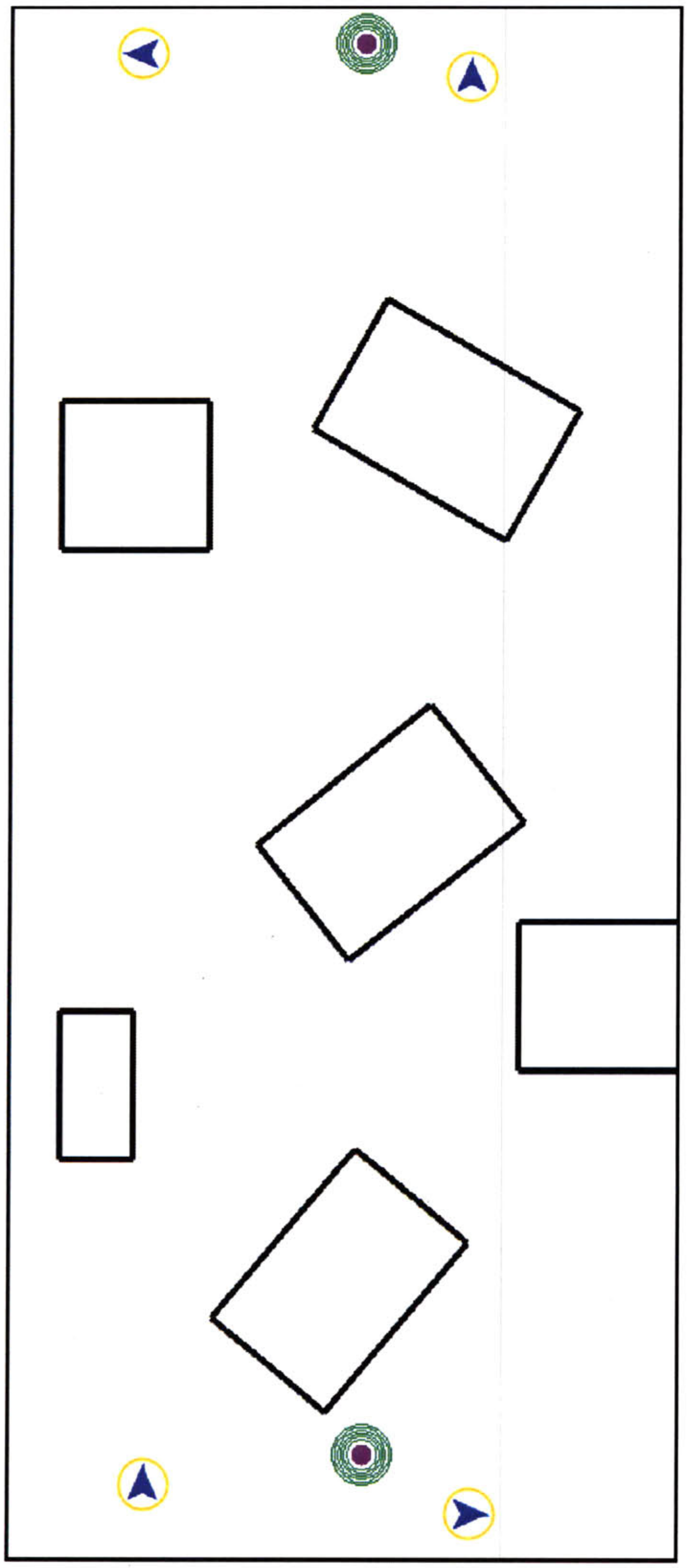


Figure 4-5: Obstacle layout, four starting positions, and two goal locations for the Four Agents with Obstacles scenario

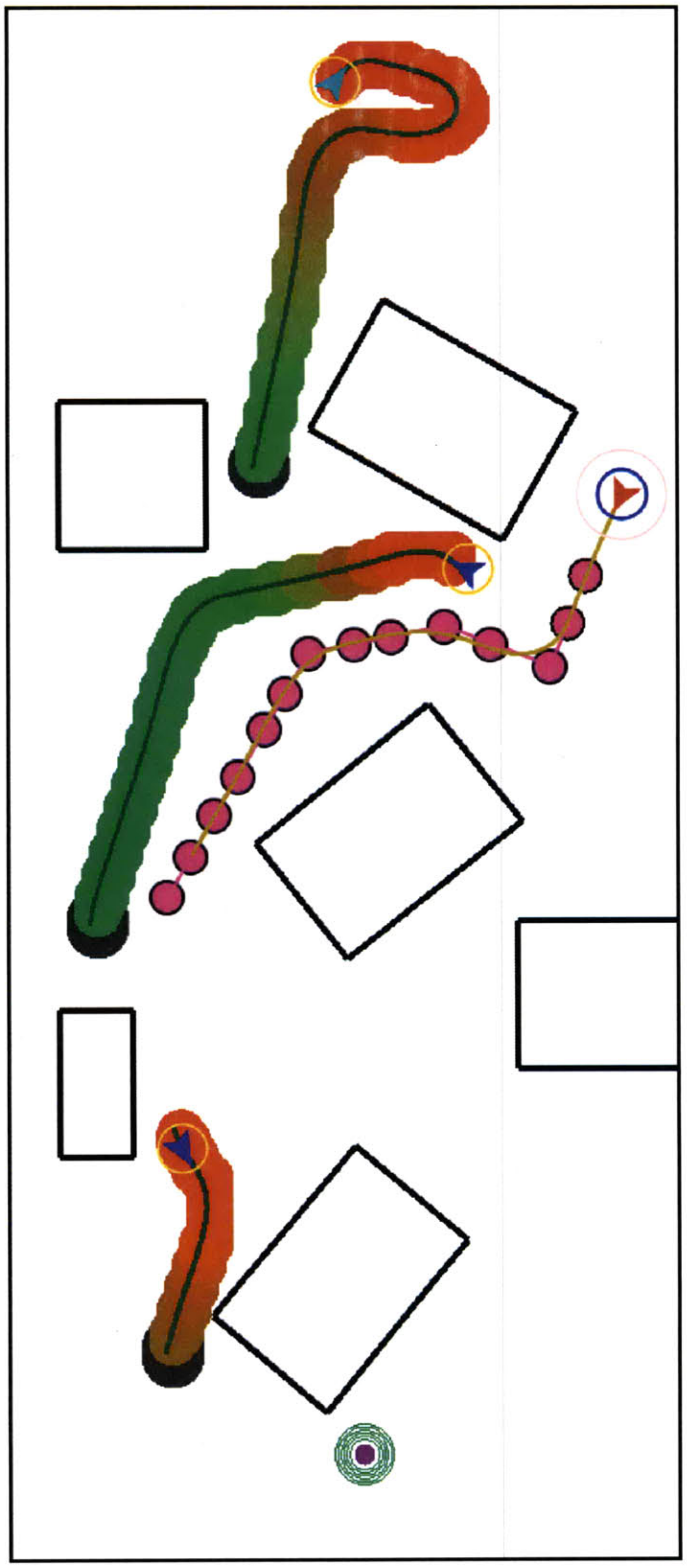
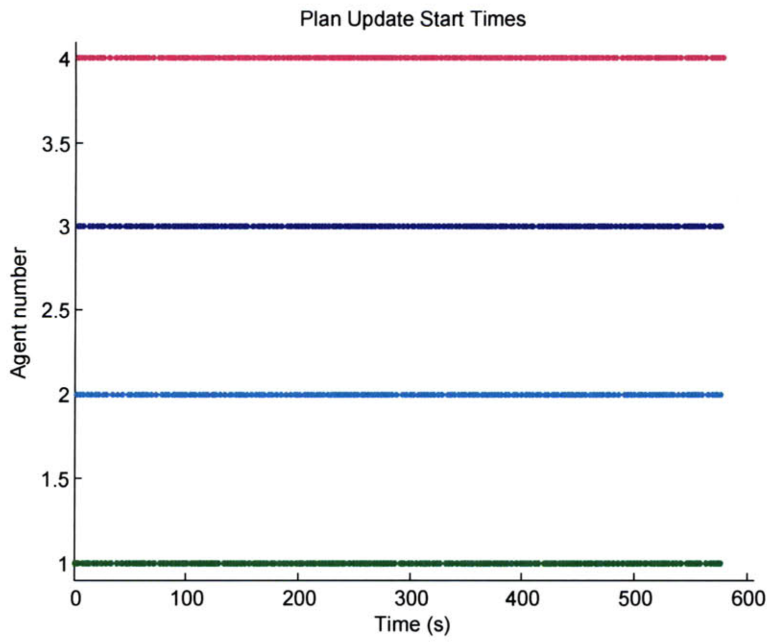
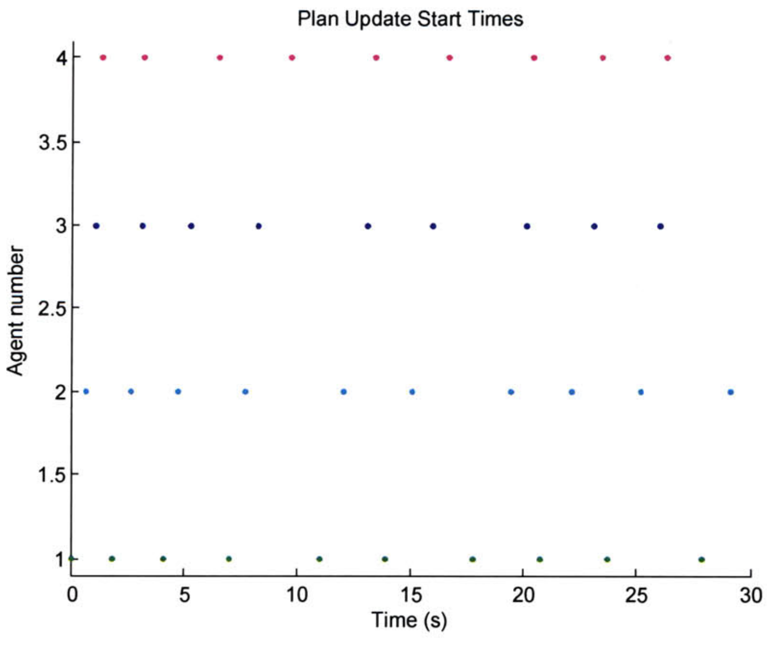


Figure 4-6: State of the world as seen by agent 1 in the Four Agents with Obstacles scenario

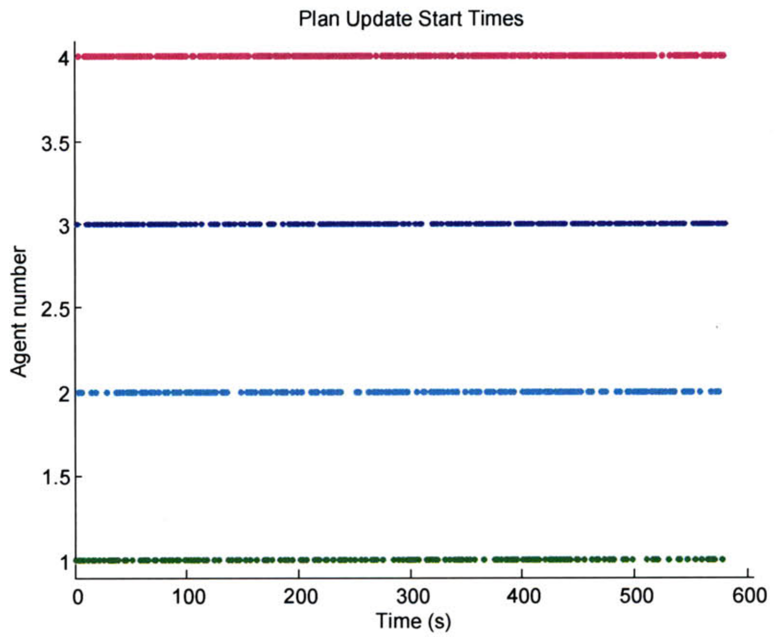


(a) Times at which each agent was allowed to update its plan during one ten-minute simulation

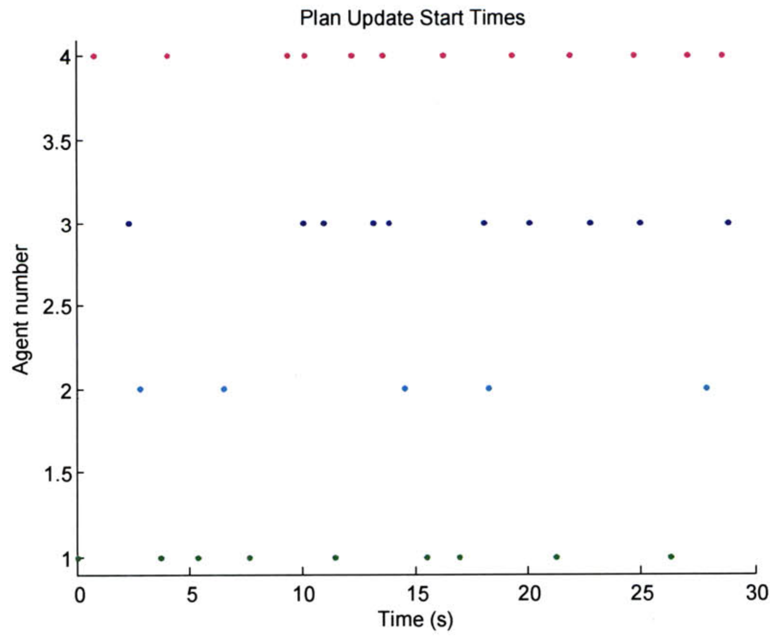


(b) Close-up showing sequential planning order

Figure 4-7: Replan start times for round-robin approach (Four Agents with Obstacles)



(a) Times at which each agent was allowed to update its plan during one ten-minute simulation



(b) Close-up showing dynamic planning order

Figure 4-8: Replan start times for merit-based token passing (Four Agents with Obstacles)

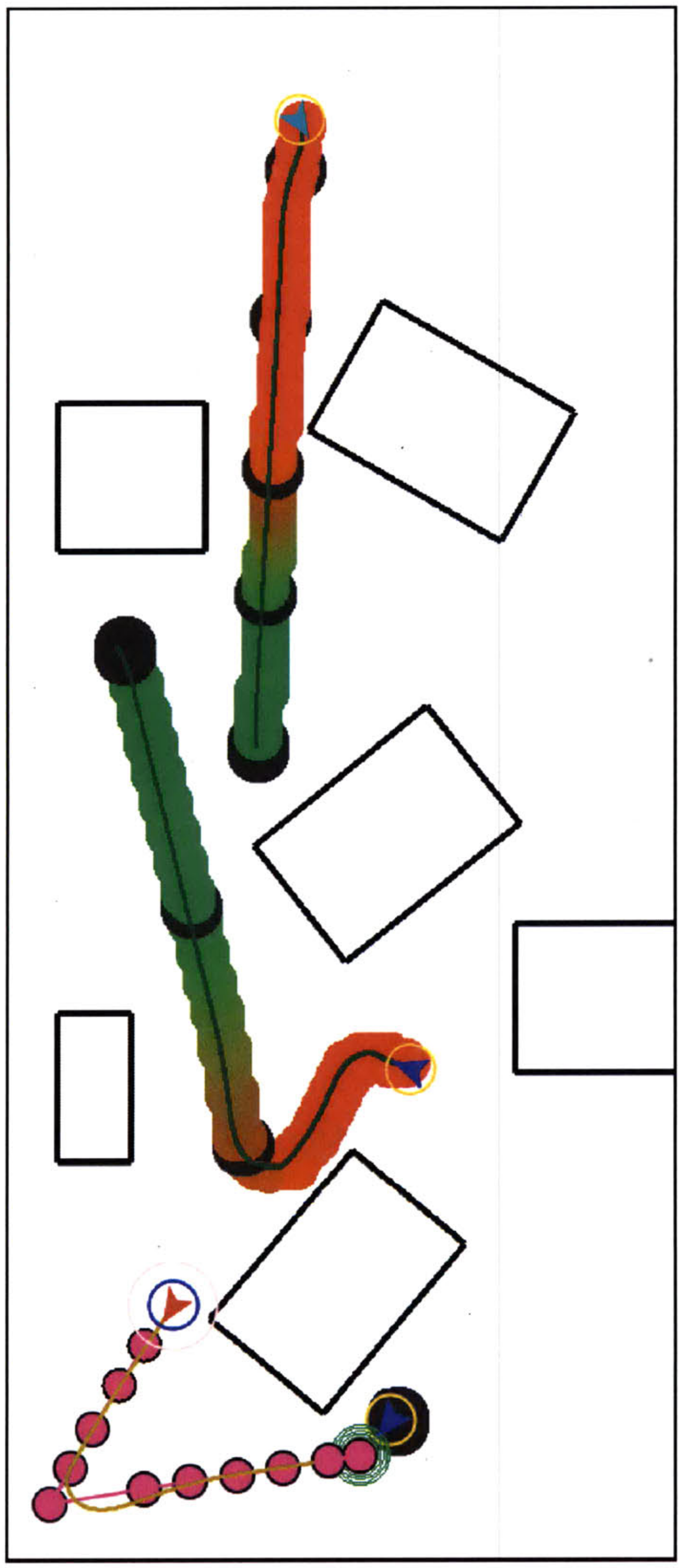


Figure 4-9: State of the world as seen by agent 1 with emergency stops enabled in the four-agent scenario. Gray areas along the other agents' trajectories indicate emergency stop nodes.

Analysis

For a team of only four agents, the merit-based token passing strategy is not expected to provide a significant improvement in performance. This can be seen from a comparison of Figure 4-7 and Figure 4-8. Even with the dynamic planning order, there are so few agents that each receives the token almost as regularly as in the round-robin case. Since each agent's ability to update its path is not significantly altered, the overall performance cannot be expected to improve just by adding this token passing strategy.

However, introducing the emergency stop logic does make a difference due to the narrow passages formed by the obstacles. As in the example scenario from Chapter 3, these passages are difficult to plan through, often resulting in agents stopping in or near the openings. Therefore, an agent that is able to find a path through one of the openings can use the emergency stop cooperation strategy to stop other agents that would otherwise obstruct this path. Furthermore, a standard error analysis[2] shows the 95% confidence interval for the mean using just the DMA-RRT algorithm in this scenario is $[11.39, 12.49]$, while the 95% confidence interval for the mean using emergency stops is $[12.53, 13.47]$. The lack of overlap between these confidence intervals is a strong indicator that the emergency stops provide a statistically significant improvement in performance.

4.2 Experimental Results

4.2.1 Setup

Hardware tests of the DMA-RRT algorithm were performed in the Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) at the MIT Aerospace Controls Laboratory. The RAVEN setup (shown in Figure 4-10) is used to test planning and control algorithms with a variety of autonomous air and ground vehicles [51, 52]. Algorithms are run on dedicated vehicle computers and the resulting actuation

[2]Standard error is computed as $SE = s/\sqrt{n}$, where s is the sample standard deviation and n is the sample size. The 95% confidence interval is given by $\bar{x} \pm 1.96SE$, where $\bar{x}$ is the sample mean [50].

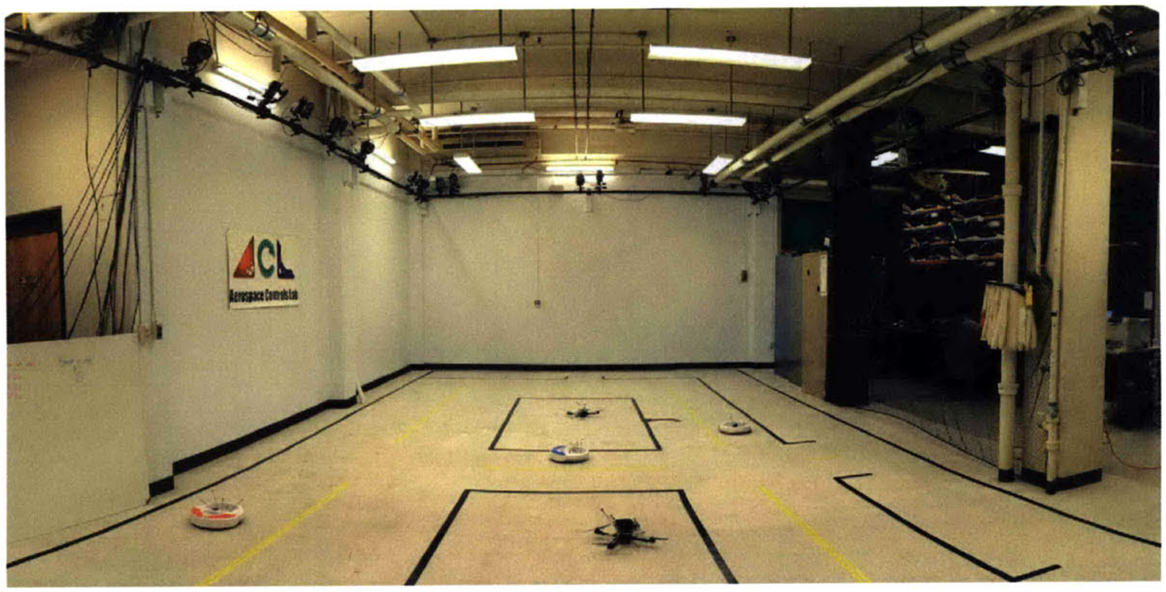


Figure 4-10: RAVEN at the Aerospace Controls Lab

commands are sent to the vehicles via a pair of xBee-PRO wireless serial modules.

Agent Tracking

The key feature of the RAVEN setup is the VICON MX-3+ motion capture system. This system uses cameras mounted along the perimeter of the test area to track vehicles operating in the visible volume. Each vehicle has a unique pattern of reflective markers affixed to it, and the cameras track these dots to provide accurate, high-bandwidth position and attitude estimates for each tracked vehicle. Additional information on the RAVEN test area is available in [51, 52]. These state estimates are then used to close the feedback loop on the vehicles for accurate closed-loop control.

Vehicle

The iRobot Create, shown in Figure 4-11, is the vehicle selected for these tests. It is a small ground robot with one drive wheel on each side and a caster in front for balance and, as such, can be modeled accurately by skid-steer dynamics (4.1). It is capable of speeds up to 0.5 m/s and has on-board speed controllers to maintain the commanded velocity. Due to their low inertia and high-torque motors, the Creates are also able to stop almost instantly. xBee-PRO modules are connected directly to these vehicles in

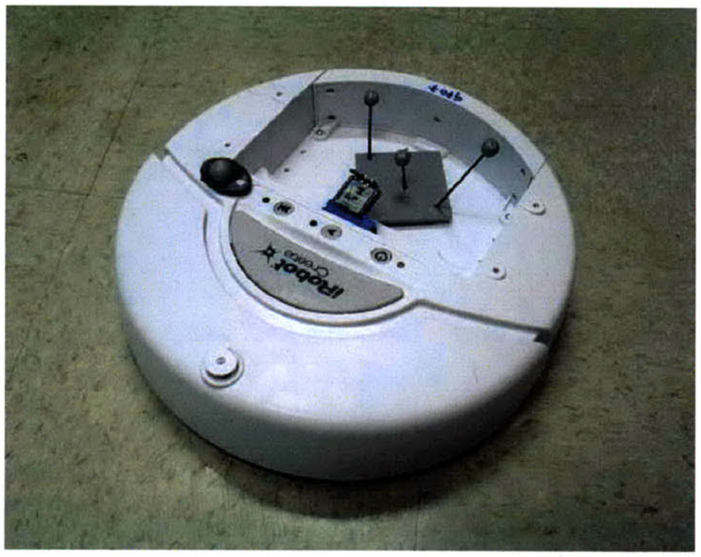


Figure 4-11: iRobot Create with xBee-PRO wireless serial module connected

order to send actuation commands.

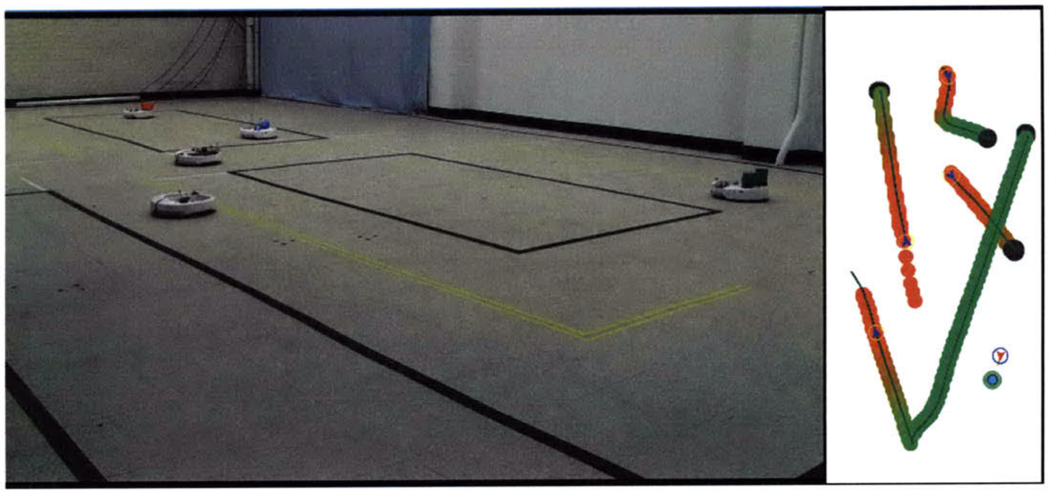
Simulations Interface

The RRT-Sim software used for the DMA-RRT simulations is designed to interface with RAVEN, allowing the user to maintain the Java implementation of the algorithm used for simulation while replacing the simulated vehicles with iRobot Creates and state feedback from VICON. Experimental results using this combination of RRT-Sim and RAVEN are presented below.

4.2.2 Results

Both the DMA-RRT and Cooperative DMA-RRT were tested in RAVEN using a team of five iRobot Creates. The open map with ten goals (Figure 4-1) was used in both cases, and the agents were started at arbitrary locations.

A snapshot from a run of the DMA-RRT algorithm is shown in Figure 4-12. As in the simulation runs, the color gradients on the trajectory visualization indicate the time at which that area of the environment is expected to be occupied by a given agent. The overlay of this visualization onto the the image shows the physical locations corresponding to these time-parameterized obstacles.


(a) Snapshot of the vehicles and their trajectories

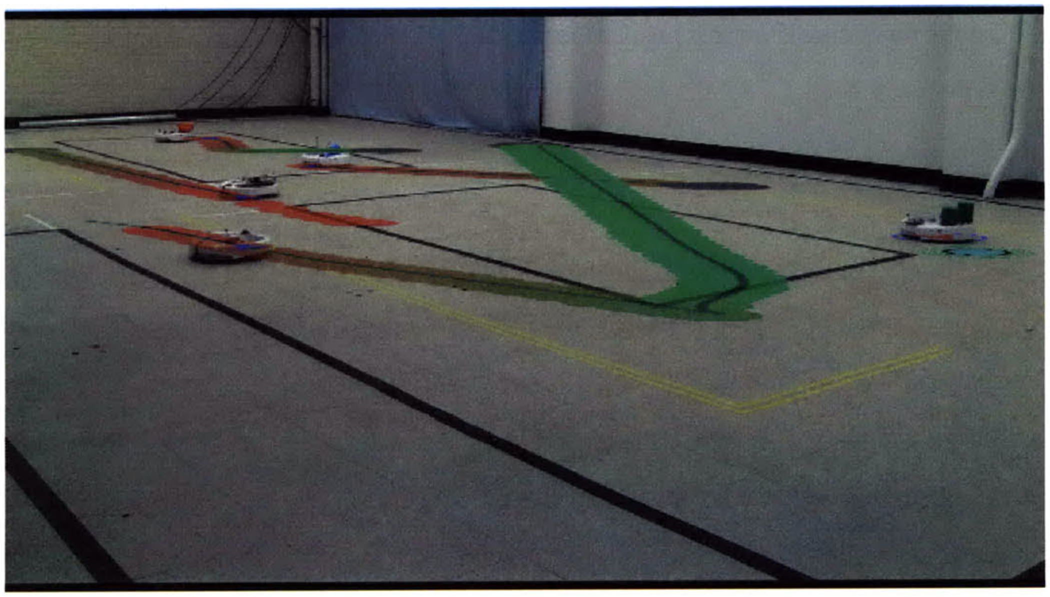

(b) Overlay of the trajectory visualization on the actual testbed

Figure 4-12: DMA-RRT in RAVEN

Table 4.1: Minimum separation distance (m) between agent pairs (DMA-RRT)

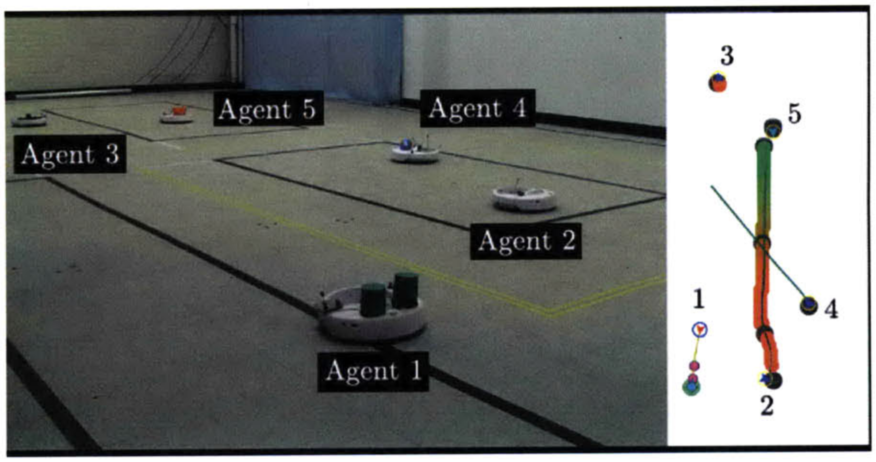
Agent	1	2	3	4
5	0.720	0.554	0.436	0.732
4	0.586	0.417	0.476	-
3	0.365	0.505	-	-
2	0.509	-	-	-

Table 4.2: Minimum separation distance (m) between agent pairs (Cooperative DMA-RRT)

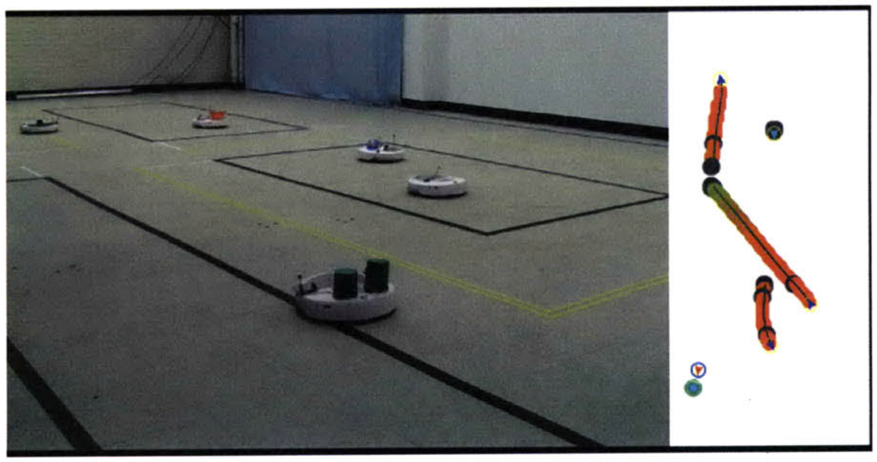
Agent	1	2	3	4
5	0.343	0.344	0.528	0.312
4	0.564	0.641	0.441	-
3	0.313	0.308	-	-
2	0.565	-	-	-

Figure 4-13 shows one instance of the emergency stop logic activating. Agent 2 first plans a path terminating just short of its goal due to position of agent 5. Agent 4 then receives the token and finds a path to its goal, but identifies a conflict with agent 2's path. Since paths that reach the goal are preferred, agent 4 sends an emergency stop message to agent 2 to terminate its path at its first stop node. Agent 2 complies and upon receiving the token from agent 4, finds an updated plan that allows it to stop closer to its goal without interfering with agent 4's path. The agents continue along these paths, with agent 2 waiting at its terminal state for agent 4 to pass.

Table 4.1 shows the minimum separation distance between agents over the course of a five minute run of the DMA-RRT algorithm. The distance between two agents placed side by side is measured via the VICON system to be 0.30 m. The minimum separation between each pair of agents is greater than this threshold for all combinations of agents, indicating that the collision avoidance constraint is satisfied for the entire run. Table 4.2 shows similar results for Cooperative DMA-RRT.



(a) Agent 2 plans a path, then agent 4 find a path to goal that conflicts with agent 2


(b) Agent 4 sends an emergency stop message to agent 2, agent 2 terminates its path early, and agent 4 is able to proceed to the goal

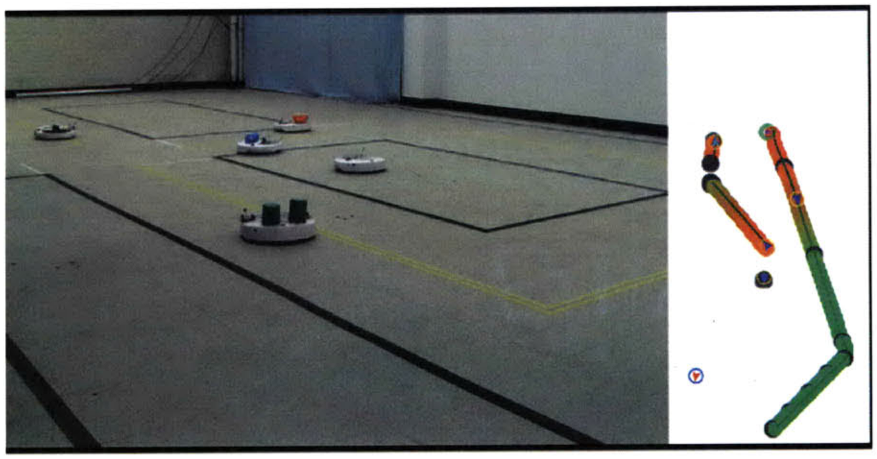

(c) Agent 2 reaches the end of its plan and waits for agent 4 to pass

Figure 4-13: Emergency stop logic in action

4.3 Summary

The simulation results presented in this chapter demonstrate that the DMA-RRT algorithm and the Cooperative DMA-RRT algorithm are able to handle large teams and complex environments. The performance improvement between the round-robin approach and merit-based token passing is evident from these trials, as is the benefit of introducing the cooperation strategy. The experimental results presents here verify that the collision avoidance requirement is met. Moreover, they demonstrate that the DMA-RRT algorithm is a viable path planner for real-world multi-agent systems. Future testing will expand these results to more complex environments and an even larger team of agents.

Chapter 5

Rover Test-Platform

The work presented thus far has been motivated by the increasing use of autonomous multi-agent systems. The Agile Robotics for Logistics project at the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) is one such example [53]. This project addresses the general problem of automating warehouses such as those of the military supply chain. These warehouses are dynamic, human-occupied, minimally structured environments, often with uneven terrain. Thus, fast, online algorithms are required to be able to interact with and adapt to these complex environments. To this end, the project aims to develop robotic systems that are able to operate safely and efficiently in areas such as a military Supply Support Activity (SSA) alongside normal personnel. The initial stage of the project focused on the development of an autonomous forklift capable of manipulating palletized cargo with minimal human supervision [53]. This chapter presents a small-scale rover that has been developed to assist the forklift in completing its tasks by serving as a mobile sensor package, while also performing auxiliary tasks, such as inventory management. As depicted in Figure 1-1, several of these rovers may be deployed alongside autonomous forklifts to perform complex cooperative tasks. For example, a forklift may summon two rovers to act as spotters while unloading a pallet from a truck in a confined space. The prerequisite to this functionality is the ability to navigate autonomously and safely in the same environments as the forklifts. To that end, the planning and control algorithms used on the forklift are adapted to this smaller robot. This functionality lays the foundation

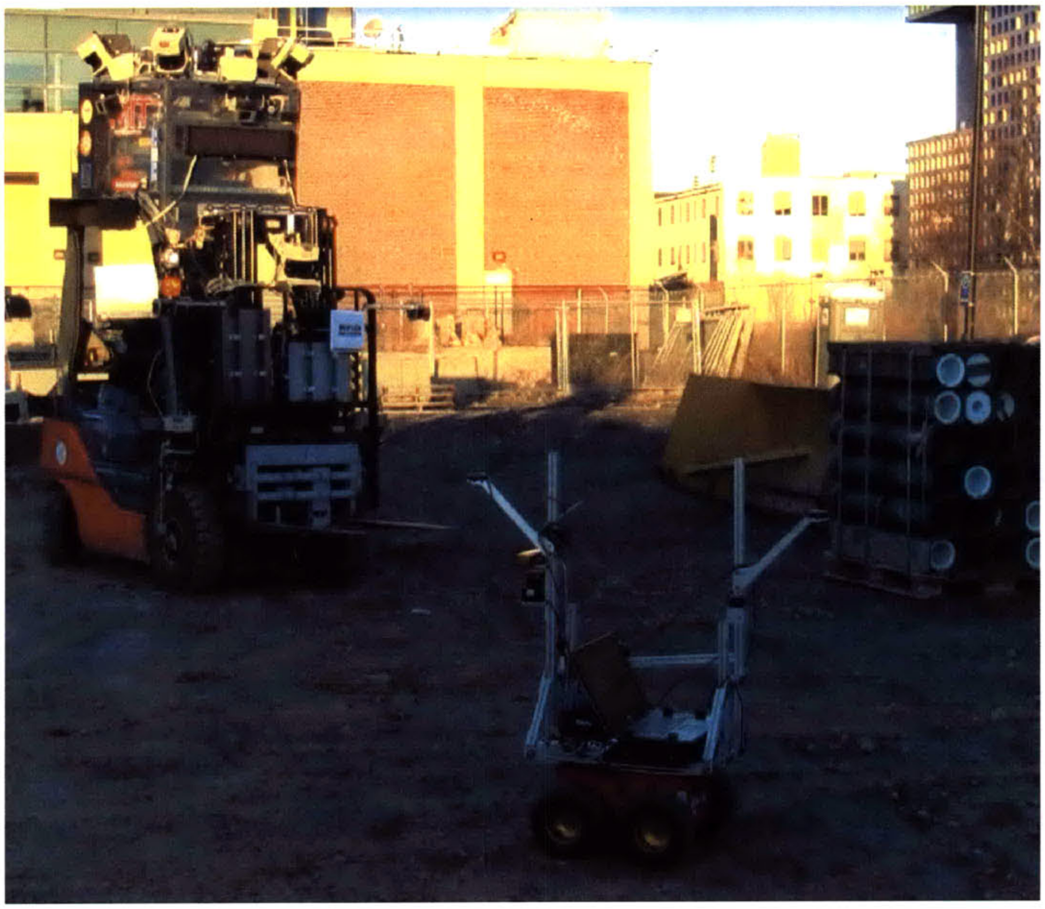


Figure 5-1: Agile Robotics forklift and rover

for a team of agents consisting of multiple forklifts and rovers — an ideal candidate for future implementation of the DMA-RRT algorithm.

5.1 Hardware

In comparison to the forklift, the rover is a relatively simple robot comprised of three main hardware components. A Pioneer 3 wheeled robot from MobileRobots forms the base of the rover, and the two wheels on each side of the robot base are coupled, mimicking a two-wheeled robot with skid-steer dynamics. The base has internal speed controllers for each side that allow it to track left and right wheel velocity commands. While the Pioneer 3 is capable of traveling at speeds over 0.7 m/s, velocity command tracking performance degrades quickly above this speed. As such, 0.7 m/s is selected as the maximum velocity command for each set of wheels. Although this maximum velocity is relatively low, the high-torque motors in the base make it a highly maneuverable, all-terrain platform well suited to the SSA environment.

The rover is also equipped with a basic sensor suite. The base robot provides velocity feedback from encoders mounted on the wheels, while an Xsens MTi inertial measurement unit (IMU) provides linear acceleration, angular rate, and attitude information. This data is used to estimate the rover's position in a local frame via dead-reckoning, and GPS measurements from a Garmin 18X-5Hz GPS receiver then map this *local frame* [54] into global coordinates. In addition, two laser range finders, Hokuyo UTM-30LX and URG-04LX, are mounted on the rover to detect obstacles in the environment. These components are illustrated in Figure 5-3.

All the sensor data processing and path planning is performed on a 3.06 GHz dual-core laptop that is mounted directly on the rover base. The resulting left and right velocity commands are sent to the rover base via a serial connection using the MobileRobots Advanced Robotics Interface for Applications (ARIA) software library [55]. The laptop supplies power to the IMU and GPS units, while an extra battery powers the laser range finders.

The fully assembled rover measures 1.26m x 0.52m x 1.22m and is shown in Figure 5-2.

5.2 Software

One of the auxiliary purposes of the rover is to demonstrate the applicability of the Agile Robotics software to robotics work in general. Just as much of the core software for the forklift is adapted from MIT's entry in the DARPA Urban Challenge [56, 57], the core software for the rover is adapted from the forklift. This is coupled with the ARIA library to interface the Agile Robotics software and the Pioneer 3 base of the rover. Then only a few changes must be made to the software for it to function correctly on the rover.

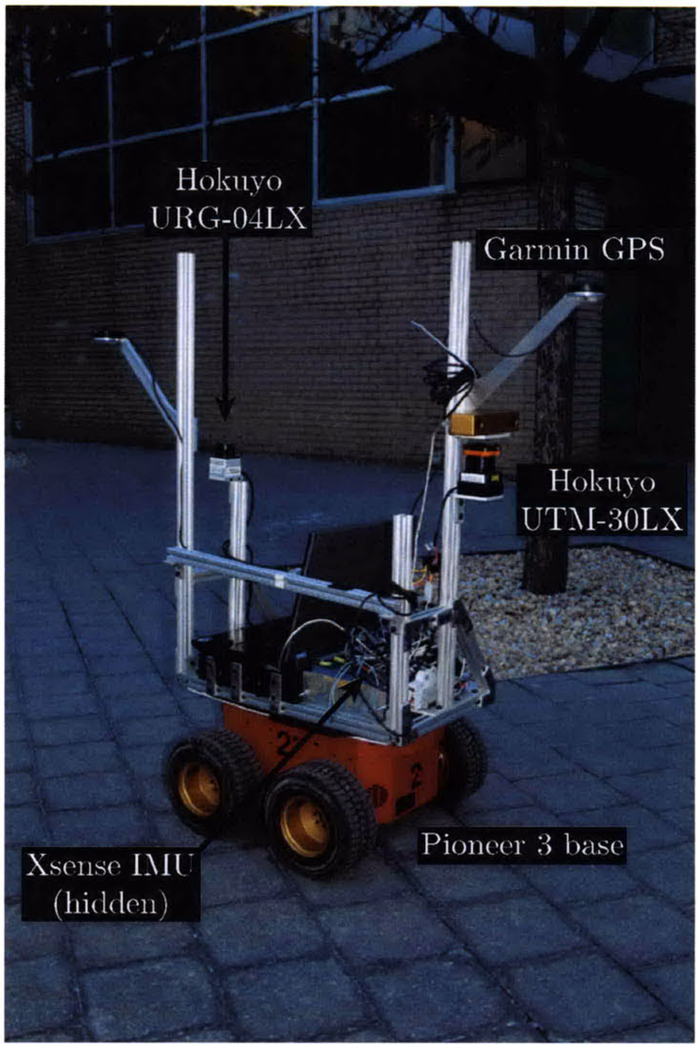


Figure 5-2: Rover with all components mounted

(a) Pioneer 3 robot used as rover base

(b) Xsens MTi IMU — 0.05° angular resolution

(c) Garmin 18X-5Hz GPS Receiver — 15m resolution, 5Hz update rate

(d) UTM-30LX — 30m range, 270° FOV, 0.25° angular resolution, 40Hz update rate

(e) URG-04LX — 4m range, 240° FOV, 0.36° angular resolution, 10Hz update rate

Figure 5-3: Rover hardware components

5.2.1 Controller

To generate actuation commands, in the form of steering, gas, and brake commands, the forklift uses a pure-pursuit steering controller and a Proportional-Integral (PI) velocity controller. The pure-pursuit steering controller is retained to track the reference paths generated by the path planner. It is adjusted for skid-steer dynamics as described in Appendix A, yielding the control law in (4.2), reproduced below:

$$\Delta v = -v\left(\frac{L_w \sin\eta}{\frac{L_1}{2} + l_a \cos\eta}\right)$$

Again, L_1 is the pure-pursuit look-ahead distance, l_a is the distance between the anchor point and the axle, and η is the angle from the vehicle to the look-ahead point on the reference path.

The smooth trajectories that arise from a pure-pursuit controller are well-suited to vehicles with turn radius constraints following long reference paths. In scenarios where humans and robots are working in the same environment, these smooth trajectories also make it easier for the humans to anticipate the behavior of the robots and be comfortable working near them. Thus the pure-pursuit controller is a suitable choice for the rover in general. However, the rover's ability to pivot in place, due to its skid-steer design, can be beneficial in certain scenarios. In particular, allowing the rover to turn sharply at low speeds can help it avoid some of the looping trajectories often observed in implementations of pure-pursuit controllers. Though this seems contradictory to the nature of the pure-pursuit controller, no additional control strategy is required; the pure-pursuit controller is capable of producing this type of behavior for appropriately selected parameters.

The minimum L_1 distance for the rover is set to 0.01m, and this extremely small minimum L_1 is part of what allows the rover to make sharp turns at low speeds.

Due to the Pioneer 3's on-board speed controllers, no additional velocity controller is needed for the rover. A speed designer (described below) generates velocity commands v_{cmd}, which when combined with the turn command Δv, can be translated into left and right velocity commands for the skid-steer rover as follows:

$$
\begin{aligned}
v_L &= v_{cmd} + \frac{\Delta v}{2} \\
v_R &= v_{cmd} - \frac{\Delta v}{2}
\end{aligned} \tag{5.1}
$$

5.2.2 Path Planner

The forklift uses the CL-RRT algorithm to generate its reference trajectories, expanding its tree using extended bicycle model dynamics [43]. This planner is almost directly translated to the rover, as the rover's skid-steer dynamics can also be represented with an extended version of the modified bicycle model (4.1), as given below:

$$
\begin{aligned}
\dot{x} &= v \cos\theta \\
\dot{y} &= v \sin\theta \\
\dot{\theta} &= \frac{\Delta v}{L_w} \\
\dot{v} &= a \\
v &= \frac{v_L + v_R}{2} \\
\Delta v &= v_L - v_R \\
|a| &\leq a_{max} \\
|\Delta v| &\leq \Delta v_{max} \\
|v_L| &\leq v_{max} \\
|v_R| &\leq v_{max}
\end{aligned} \tag{5.2}
$$

The control inputs to the rover are given by v_L and v_R. Rotation rate can be bounded artificially by selecting $\Delta v_{max} < 2v_{max}$, and a value of 0.3 m/s was found to allow the rover to turn at a reasonable speed while avoiding jittery motion observed during fast turns. The track L_w (distance between left and right wheels) is measured to be 0.40m, but a scale factor must be added to account for the pair of coupled wheels on each side [49]. The resulting effective L_w for the rover is 0.58m.

During the tree growing process, a speed designer is used to generate velocity commands to ramp up from rest and ramp down to stop at the end of its trajectory. These velocity commands are used to compute the pure-pursuit steering commands for the closed-loop simulation. Since the pure-pursuit L_1 distance increases with the velocity command, low velocity commands will produce small L_1 distances, allowing the rover to make sharper turns at low speeds. While this is an inherent property of the controller, the effect is exaggerated on the rover by reducing the minimum L_1 distance severely and introducing an attenuation factor on the velocity commands that is a function of the previous turn command as given below:

$$v_{cmd} = \tilde{v}_{cmd}(1 - \frac{|\Delta v|}{\Delta v_{max}}) \tag{5.3}$$

where $\tilde{v}_{cmd}$ is the output of the original speed designer adapted from the forklift.

The net result is such that if the rover generates a sharp turn command, i.e., a large Δv, while it is moving slowly, this attenuation factor will prevent v_{cmd} from ramping up quickly. Maintaining this lower speed keeps L_1 small, which allows the rover to finish making the sharp turn. As the rover approaches its desired heading angle, the turn commands naturally decrease, allowing the speed command to ramp up again. Once at higher speeds, the larger L_1 distance prevents large Δv commands, reducing the effects of the attenuation term.

Table 5.1 presents a summary of the parameters used in the rover controller and planner.

Table 5.1: Rover controller and planner parameters

L	0.28 m
L_1	0.01 m - 1.00 m
l_a	0.01 m
L_w	0.58 m
a_{max}	0.87 m/s^2
v_{max}	0.7 m/s
Δv_{max}	0.3 m/s

5.3 Experimental Results

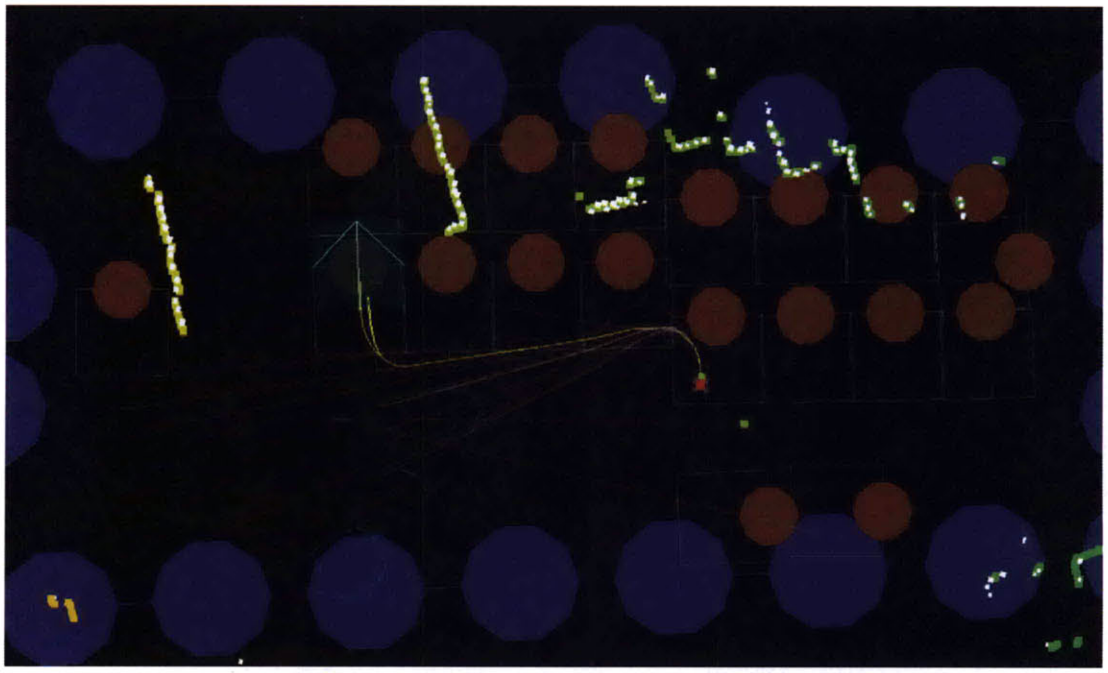
5.3.1 Outdoor Navigation

The rover's CL-RRT and controller were tested in several outdoor environments, including the SSA in Figure 5-4 at Fort Lee, VA.

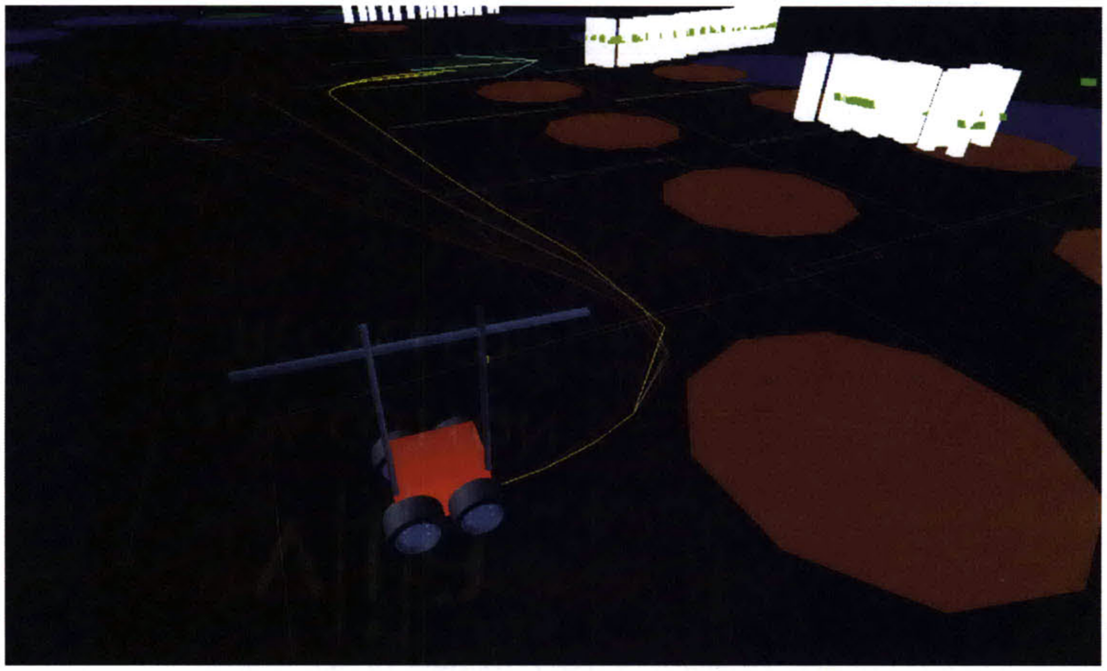
Figure 5-4: Outdoor Supply Support Activity (SSA) at Ft. Lee, VA

The rover's knowledge of the world from one example run is shown in Figure 5-5. The large blue dots mark the perimeter, orange dots are potential goal locations (storage bays, loading areas, etc), and the turquoise square is the current goal. The perimeter and potential goal locations are defined in a pre-specified topological map of GPS positions. All objects detected by the LIDAR are shows as green dots, and the white pillars are the obstacles inferred from this data. The rover's selected path is shown in yellow, surrounded by several other potential paths in the tree.

The rover plans in its local frame, avoiding all LIDAR-based obstacles. In addition, it performs a coordinate transform using GPS data to map the goal and perimeter information into this frame. Since the rover was operating in a relatively open part of the SSA and started far from the goal, it selected a smooth path to the goal, as expected from pure-pursuit.



(a) Birds-eye view of the SSA showing the RNDF, LIDAR obstacles, and CL-RRT plan



(b) 3D visualization of the rover and obstacles

Figure 5-5: Outdoor navigation to a goal defined in GPS coordinates

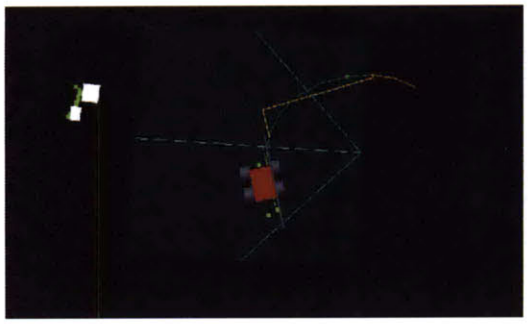
5.3.2 Sharp Turn

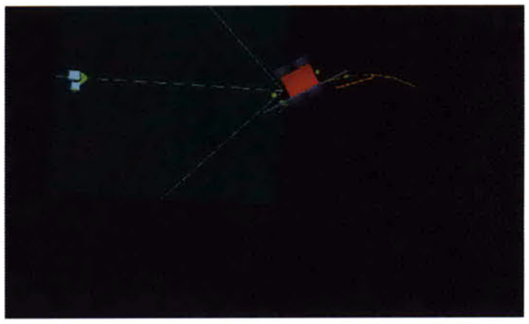
The ability to perform sharp turns at low speeds can be beneficial in both indoor and outdoor environments. Constrained indoor environments are often difficult to plan smooth paths through, while GPS jumps can lead to sudden changes in the goal location in outdoor environments. The example in Figure 5-6 illustrates the GPS jump scenario.

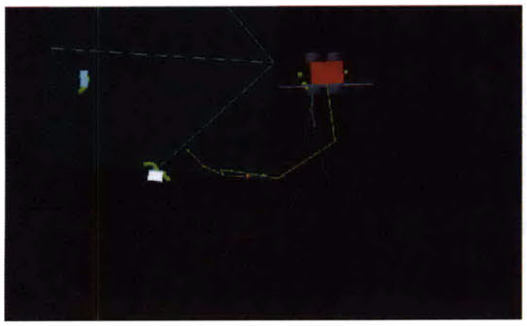
The rover's original path is designed to terminate inside the goal region after reaching the heading angle specified for that goal. However, a sudden jump in the GPS data moves the goal such that the original path no longer ends inside it. The rover comes to a stop and grows the CL-RRT to identify a new path. Since the rover is currently at rest and its new path leaves departs at a very different angle, it is able to issue the maximum turn command to essentially pivot in place. Once it is aligned with the new path, it increases its velocity and proceeds along the path to reach the goal.

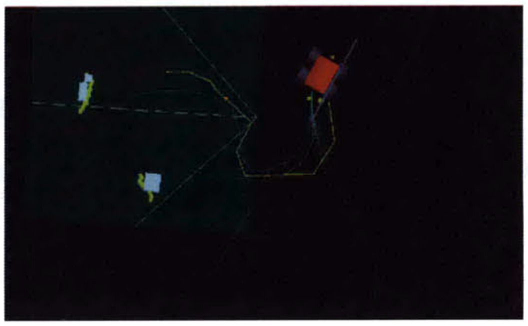
5.3.3 Summary

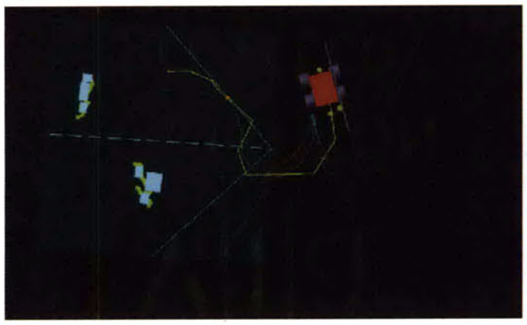
The small scale rover presented in this chapter is a key component in the expansion of the Agile Robotics project to coordinated, multi-agent operations. The rover's basic sensor suit allows it to localize itself relative to GPS data while also detecting obstacles in the world. The CL-RRT implementation on the rover makes full use of this capability to navigate in the types of complex environments found in military supply areas. Moreover, this basic planning capability lays the foundation for the DMA-RRT algorithm to be implemented on a team of rovers and forklifts, allowing them to perform their tasks safely and efficiently.

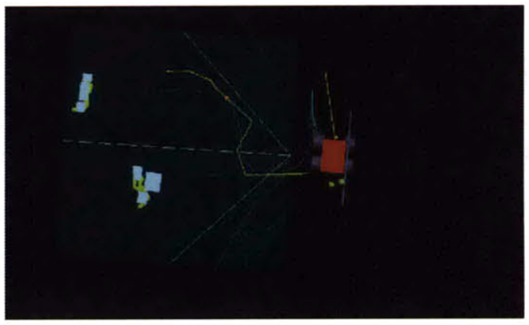

(a) Original trajectory to stop in goal with desired heading (indicated by the arrow)

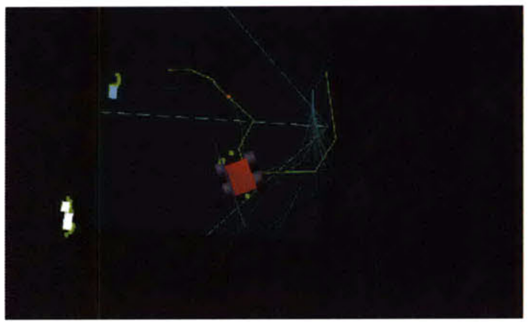

(b) Goal moves due to GPS jump


(c) Rover stops and computes new path


(d) Begins sharp turn to align with new path


(e) Completes turn


(f) Follows new path


(g) Continues along path into goal area

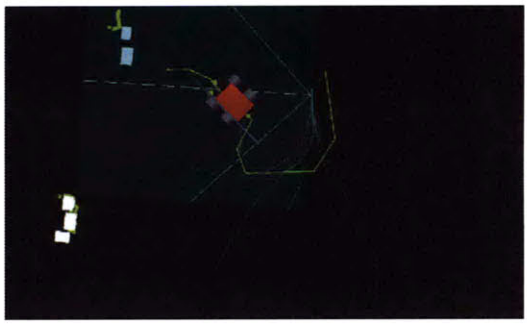

(h) Reverses into goal with desired heading

Figure 5-6: Rover executing a sharp turn to reach goal after GPS jump

Chapter 6

Conclusion

6.1 Summary

This thesis addresses the problem of path planning for cooperative multi-agent autonomous systems operating in complex environments. As teams of autonomous agents are deployed in increasingly challenging situations, the need for all agents in the team to navigate safely and efficiently becomes crucial. To this end, this thesis proposes the Decentralized Multi-Agent Rapidly-exploring Random Tree (DMA-RRT) algorithm. This algorithm combines the Closed-loop RRT path planner with a new coordination strategy, merit-based token passing, to improve team performance while ensuring all constraints (dynamics, environment, inter-agent) are satisfied.

Chapter 2 provides an overview of the Closed-loop RRT (CL-RRT) algorithm as a core component of the DMA-RRT algorithm. The use of CL-RRT as the baseline planner is motivated by its success as a fast, online path planner that is able to handle the type of complexity found in practical systems, such as constrained, nonlinear agent dynamics as well as cluttered or restrictive environments. Its sample-based approach enables it to quickly explore these environments, while the closed-loop simulation used to grow the tree accounts for the complexity in the dynamics and environment to produce feasible paths. The tree of feasible paths grown and maintained by the CL-RRT contains valuable information about an agent's potential plans. Thus, the merit-based token passing strategy is developed to use this information to coordinate a team of agents in which each agent plans using the CL-RRT algorithm. By comparing the cost of its current path to the cost of the best feasible path in its tree, each agent is able to measure its incentive to replan, i.e. its Potential Path Improvement (PPI), at any time. This allows the agents in the team to plan in a dynamic sequence where agents bid to plan next according to their PPI. As a result, agents with more to gain from replanning are able to do so more often, while agents with little reason to replan do not disrupt the planning order. This improves the overall planning efficiency of the team, which, in turn, improves overall performance. The DMA-RRT algorithm encapsulates these two components and adds waypoint communication between agents. From these waypoints, each agent is able to determine the other agents' current trajectories and select plans that satisfy all inter-agent constraints. The dynamic sequential planning order combined with this waypoint sharing guarantees that all inter-agent constraints will be satisfied for all time. Furthermore, the low communication requirements for this waypoint sharing and the use of an individual CL-RRT on each agent allows the DMA-RRT algorithm to scale to large teams.

In Chapter 3, an additional cooperation component is introduced to the DMA-RRT algorithm. In the normal DMA-RRT algorithm, each agent selects its paths solely to minimize its local cost function. However, this individually greedy approach does not necessarily minimize the global cost for the entire team. The Cooperative DMA-RRT algorithm introduces pairwise cooperation between agents in the form of emergency stop node checks. Agents identify nodes in their path at which they could stop indefinitely, if needed, and the additional cost associated with stopping at these nodes. This information is communicated to the team as part of the usual DMA-RRT waypoint communication. Then when an agent replans, it gains the option to truncate another agent's path at one of these emergency stop nodes if doing so enables the replanning agent to select a far better path, thus reducing the combined cost for both agents.

Chapter 4 verifies the properties of the DMA-RRT and Cooperative DMA-RRT algorithms through a series of simulations and experimental results. True to the nature of the algorithm, the simulations are performed across multiple computers on a common network. The results show a significant performance improvement with the merit-based token passing strategy, as opposed to a round-robin planning order. Similarly, adding the emergency stop cooperation logic showed a significant performance improvement over the normal DMA-RRT algorithm. The various team sizes used in the simulations and hardware tests, particularly the ten-agent scenario, demonstrate the algorithms' scalability. The experimental results also verify that the collision avoidance inter-agent constraint is satisfied, and moreover, show that the DMA-RRT algorithm is a viable path planner for real multi-agent systems.

Finally, Chapter 5 introduces a rover test platform being developed as part of the Agile Robotics project. A team of these rovers is envisioned as a cooperative multi-agent team capable of assisting an autonomous forklift and performing unique tasks, such as inventory management. The software architecture for the Agile Robotics project is designed to be applicable to general robotic systems, and as such, only minor modifications need to be made to translate this functionality to the rover. A series of experimental results demonstrate the rover's ability to plan using the CL-RRT algorithm in real-world environments, such as an outdoor warehouse scenario with rough terrain and LIDAR-based obstacles. This provides the core functionality needed to implement the DMA-RRT algorithm on a team of rovers.

6.2 Future Work

There are many interesting topics for future research that will help extend the work presented here to a more general class of multi-agent path planning problems. A few key topics are discussed in this section.

The DMA-RRT algorithm is developed under the assumption of a static environment (Section 2.1.3). However in practice, agents will typically be operating in highly dynamic environments. Furthermore, the CL-RRT algorithm is designed to handle dynamic obstacles via the lazy check. Section 3.4 describes one potential approach where emergency stop nodes are placed at every possible node along an agent's path. This would allow the agent to stop safely anywhere on the path, either due to a termination message received from another agent or in response to a dynamic obstacle detected via additional sensors. However, this conservative approach requires each agent to claim a potentially large portion of the state space, adding largely unnecessary complexity to the planning problem in the form of frequent emergency stop node checks.

Relaxing the assumption in Section 2.1.1 that the agents form a fully connected network, for example by adding communication range limits, introduces the concept of subnetworks. Consider an agent's subnetwork to be the set of all agents within its communication radius. While each subnetwork on its own can be treated as a fully connected network for the DMA-RRT algorithm, an additional framework is needed to manage interactions and overlaps between subnetworks. The the constraint satisfaction guarantee must also be revised due to the uncertainty in the environment beyond the communication radius.

The problem of managing agents as they move between subnetworks can be generalized to the problem of token creation and rejection. For example, in the event that the token is lost, perhaps due to the token holder leaving the network, agents must be able to create a new token. However, multiple agents in the same subnetwork must not be able to generate tokens at the same time. Likewise, if an agent were to introduce an extra token into a subnetwork, it must be rejected quickly and the offending agent's plan must be accounted for to prevent constraint violations. Token management algorithms [58] from the computer networking community may be especially useful here.

As described in Chapter 5, implementing the CL-RRT algorithm on the Agile Robotics rover is only the first step in that project. The goal is to have multiple rovers cooperating with each other as well as with the autonomous forklift. To transform the CL-RRT implementation on each rover into an implementation of the DMA-RRT algorithm, an inter-agent communication protocol must be established. A likely candidate for this is the Lightweight Communications and Marshalling (LCM) library [57] currently used for inter-process communication. Furthermore, the rovers and forklift typically operate on rough, uneven terrain. This introduces uncertainty in their ability to accurately follow a selected trajectory. However, as described by Luders et al. [42], bounds can be established on the tracking error for the CL-RRT algorithm. These bounds can then be incorporated into the trajectory re-simulation step of the DMA-RRT (Section 2.4.2) to generate more conservative, but safer, time-parameterized obstacles.

Appendix A

Pure-Pursuit Controller

The pure-pursuit steering controller has been used for many ground and air vehicle applications [59, 60]. Kuwata et al. [23] describe the form of the controller considered here and provide additional stability analysis. The overall goal of the controller is to track a reference path by adjusting the steering angle to compensate for the error in heading between the vehicle and a lookahead point located on the reference path. The adaptation of this control law to skid-steer vehicle dynamics is described in this appendix.

First, consider the basic kinematic bicycle model described by

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan(\delta) \end{aligned} \tag{A.1}$$

where x and y specify the position of the rear axle and θ is the vehicle's heading angle relative to the x-axis. The control inputs are velocity v and and steering angle δ. The parameter L is the distance between front and rear axles (wheelbase).

The pure-pursuit steering controller is given by

$$\delta = -\tan\left(\frac{L\sin\eta}{\frac{L_1}{2} + l_a\cos\eta}\right) \tag{A.2}$$

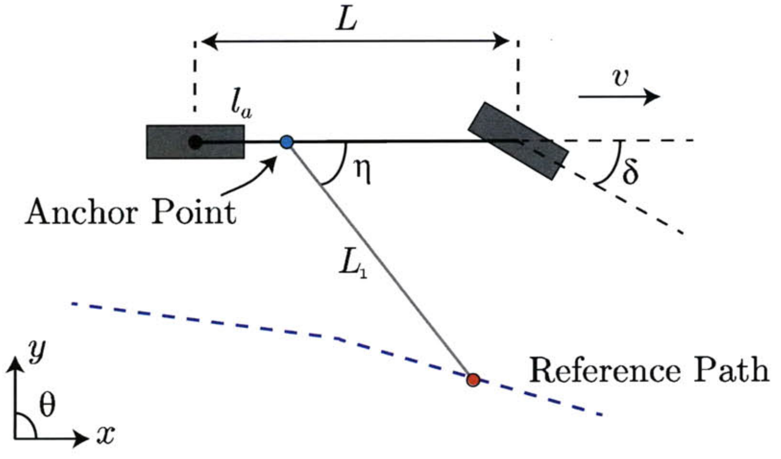


Figure A-1: Bicycle model with pure pursuit parameters labeled

where L_1 is the lookahead distance that determines the position of the lookahead point on the reference path, η is the angle between the vehicle and the lookahead point, and l_a is the anchor point distance from the rear axle, used to provide added stability. The steering command is computed such that the anchor point is steered to the reference path. These parameters are illustrated on a bicycle model in Figure A-1.

Substituting this control law (A.2) into the last line of (A.1) and simplifying yields $\dot{\theta}$ as a function of η

$$\dot{\theta} = -v\left(\frac{\sin\eta}{\frac{L_1}{2} + l_a\cos\eta}\right) \tag{A.3}$$

Note that the wheelbase term L drops out.

The modified bicycle model for skid-steer vehicles is

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{\Delta v}{L_w} \end{aligned} \tag{A.4}$$

where $v = \frac{1}{2}(v_L + v_R)$, the average of the left and right wheel velocities, $\Delta v = v_L - v_R$, the difference between the left and right wheel velocities, and L_w is the distance between the left and right wheels. Since there is only one axle, the wheelbase L is assumed to be zero.

Since the objective is to steer the skid-steer model as if it were a standard bicycle model, the $\dot{\theta}$ equations can be equated to produce a relationship between Δv and η

$$\Delta v = -v \left(\frac{L_w \sin \eta}{\frac{L_1}{2} + l_a \cos \eta} \right) \tag{A.5}$$

This provides a mapping between the pure pursuit heading term η and the left and right velocity commands issues to the skid steer vehicle. Furthermore, this control law does not depend on L, which does not exist for a two-wheeled skid steer vehicle, but rather on L_w.

The performance of the pure pursuit controller can tuned by appropriate selection of the lookahead distance. A detailed discussion of how to select the L_1 is given in Kuwata et al. [23], but the key property is that the L_1 distance is scheduled with the velocity command issued to the vehicle. Larger L_1 values prevent the vehicle from making sharp turns, which improves performance and stability at higher speeds. Furthermore, the velocity command is used rather than the vehicle's measured velocity in order to avoid jitter in the steering command due to measurement noise.

On a skidsteer vehicle, setting the minimum L_1 distance, corresponding to very low speeds, to be small allows the agent to provide sharp turn commands, taking advantage of the skid steer capabilities. Also, while the choice of anchor point distance $l_a > 0$ provides additional stability, on a pure pursuit vehicle, setting $l_a = 0$ is often reasonable to promote its ability to turn sharply. However, in other applications, $l_a > 0$ may be preferred to effectively "pull" the vehicle by some point in front of the axle, thus smoothing out the trajectory.

Bibliography

[1] "Unmanned aircraft systems roadmap, 2005-2030," tech. rep., Office of the Secretary of Defense, August 2005.

[2] S. M. LaValle, *Planning Algorithms.* Cambridge, U.K.: Cambridge University Press, 2006.

[3] M. Flint, M. Polycarpou, and E. Fernández-Gaucherand, "Cooperative path-planning for autonomous vehicles using dynamic programming," in *Proceedings of the IFAC World Congress*, (Barcelona, Spain), 2002.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions of Systems Science and Cybernetics*, vol. 4, pp. 100–106, July 1968.

[5] M. Likhachev and A. Stentz, "R* search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 344–350, 2008.

[6] S. Koenig and M. Likhachev, "D^* lite," in *Proceedings AAAI National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[7] M. Likhachev, *Search-based Planning for Large Dynamic Environments.* PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, September 2005.

[8] D. Bertsekas, *Dynamic Programming and Optimal Control.* Athena Scientific, 2005.

[9] T. Schouwenaars, B. de Moor, E. Feron, and J. P. How, "Mixed integer programming for multi-vehicle path planning," in *Proceedings of the European Control Conference*, (Porto, Portugal), pp. 2603–2608, European Union Control Association, September 2001.

[10] W. B. Dunbar and R. M. Murray, "Model predictive control of coordinated multi-vehicle formations," in *Proceedings of the IEEE Conference on Decision and Control*, (Las Vegas, NV), pp. 4631–4636, December 2002 2002.

[11] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice - a survey," *Automatica*, vol. 25, pp. 335–348, May 1989.

[12] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, 1985.

[14] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, pp. 224–241, March-April 1992.

[15] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE International Conference on Robotics and Automation*, pp. 1398–1404, 1991.

[16] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Transactions on Robotics & Automation*, vol. 8, pp. 501–518, Oct. 1992.

[17] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. C-32, pp. 108–120, February 1983.

[18] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, August 1996.

[19] L. K. Dale and N. M. Amato, "Probabilistic roadmaps - putting it all together," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Seoul, Korea), pp. 1940–1947, May 2001.

[20] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep. 98-11, Iowa State University, October 1998.

[21] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions: the Fourth Workshop on the Algorithmic Foundations of Robotics*, 2001.

[22] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, pp. 116–129, January-February 2002.

[23] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2008.

[24] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using RRT," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, (Nice, France), pp. 1681–1686, September 2008.

[25] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, Sept. 2009.

[26] T. Keviczky, F. Borrelli, and G. J. Balas, "A study on decentralized receding horizon control for decoupled systems," in *American Control Conference (ACC)*, (Boston, MA), pp. 4921–4926, June-July 2004.

[27] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, pp. 215–233, Jan. 2007.

[28] P. Scerri, S. Owens, B. Yu, and K. Sycara, "A decentralized approach to space deconfliction," in *Proc. 10th Int Information Fusion Conf*, pp. 1–8, 2007.

[29] O. Purwin and R. D'Andrea, "Path planning by negotiation for decentralized agents," in *American Control Conference (ACC)*, pp. 5296–5301, 9-13 July 2007.

[30] O. Purwin, R. D'Andrea, and J. Lee, "Theory and implementation of path planning by negotiation for decentralized agents," *Robotics and Autonomous Systems*, vol. 56, no. 5, pp. 422–436, 2008.

[31] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: A study in multi-agent hybrid systems," *IEEE Trans. on Automatic Control*, vol. 43, April 1998.

[32] G. Hoffmann and C. Tomlin, "Decentralized cooperative collision avoidance for acceleration constrained vehicles," in *IEEE Conference on Decision and Control (CDC)*, pp. 4357–4363, 2008.

[33] V. Desaraju, H. C. Ro, M. Yang, E. Tay, S. Roth, and D. Del Vecchio, "Partial order techniques for vehicle collision avoidance: Application to an autonomous roundabout test-bed," in *IEEE International Conference on Robotics and Automation*, pp. 82–87, 2009.

[34] G. S. Aoude, B. D. Luders, D. S. Levine, and J. P. How, "Threat-aware Path Planning in Uncertain Urban Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Taipei, Taiwan), October 2010 (to appear).

[35] G. Inalhan, D. Stipanovic, and C. Tomlin, "Decentralized optimization, with application to multiple aircraft coordination," in *Proceedings of the IEEE Conference on Decision and Control*, (Las Vegas, NV), December 2002.

[36] Y. Kuwata, A. Richards, T. Schouwenaars, and J. P. How, "Decentralized robust receding horizon control for multi-vehicle guidance," in *American Control Conference (ACC)*, pp. 2047–2052, June 2006.

[37] A. Richards and J. P. How, "A decentralized algorithm for robust constrained model predictive control," in *American Control Conference (ACC)*, (Boston, MA), June-July 2004.

[38] A. N. Venkat, J. B. Rawlings, and S. J. Wright, "Stability and optimality of distributed model predictive control," in *Proceedings of the IEEE Conference on Decision and Control*, (Seville, Spain), pp. 6680–6685, December 2005.

[39] P. Trodden and A. Richards, "Robust distributed model predictive control using tubes," in *Proceedings of the American Control Conference*, (Minneapolis, MN), pp. 2034–2039, June 2006.

[40] D. Bertsekas and R. Gallager, *Data networks*. Prentice-hall Englewood Cliffs, NJ, 1992.

[41] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, pp. 1105 – 1118, Sept. 2009.

[42] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How, "Bounds on tracking error using closed-loop rapidly-exploring random trees," in *American Control Conference (ACC)*, (Baltimore, MD, USA), June-July 2010.

[43] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2008.

[44] T. Schouwenaars, J. How, and E. Feron, "Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, (Providence, RI), August 2004.

[45] T. Schouwenaars, *Safe Trajectory Planning of Autonomous Vehicles.* PhD dissertation, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, February 2006.

[46] Y. Kuwata, A. Richards, T. Schouwenaars, and J. P. How, "Decentralized Robust Receding Horizon Control for Multi-vehicle Guidance," *American Control Conference (ACC)*, pp. 2047–2052, June 2006.

[47] Y. Kuwata, A. Richards, T. Schouwenaars, and J. P. How, "Distributed robust receding horizon control for multi-vehicle guidance," *IEEE Transactions on Control Systems Technology*, vol. 15, pp. 627–641, July 2007.

[48] D. Levine, B. Luders, and J. P. How, "Information-rich path planning with arbitrary constraint using rapidly-exploring random trees," in *AIAA Infotech@Aerospace Conference*, (Atlanta, GA), April 2010 (AIAA-2010-3360).

[49] C. Sae-Hau, "Multi-vehicle rover testbed using a new indoor positioning sensor." SM thesis draft, Massachusetts Institute of Technology, September 2003.

[50] D. Bertsekas and J. Tsitsiklis, *Introduction to probability.* Athena Scientific Belmont, Massachusetts, 2008.

[51] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron, "Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, (Keystone, CO), August 2006.

[52] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, pp. 51–64, April 2008.

[53] S. Teller, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. P. How, J. Jeon, S. Karaman, B. Luders, N. Roy, T. Sainath, and M. R. Walter, "A voice-commanded robotic forklift working alongside humans in minimally-prepared outdoor environments," in *IEEE International Conference on Robotics and Automation*, 2010.

[54] D. Moore, A. Huang, M. Walter, E. Olson, L. Fletcher, J. Leonard, and S. Teller, "Simultaneous local and global state estimation for robotic navigation," in *IEEE International Conference on Robotics and Automation*, (Kobe, Japan), pp. 3794–3799, May 2009.

[55] MobileRobots, "Mobilerobots' advanced robot interface for applications (ARIA)." http://robots.mobilerobots.com/wiki/ARIA.

[56] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[57] A. Huang, E. Olson, and D. Moore, "LCM: Lightweight communications and marshalling," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

[58] E. E. Johnson, Z. Tang, M. Balakrishnan, J. Rubio, H. Zhang, and S. S, "Robust token management for unreliable networks," in *Proc. IEEE Military Communications Conf. MILCOM 2003*, vol. 1, pp. 399–404, 2003.

[59] O. Amidi and C. Thorpe, "Integrated Mobile Robot Control," in *Proceedings of SPIE* (W. H. Chun and W. J. Wolfe, eds.), vol. 1388, (Boston, MA), pp. 504–523, SPIE, Mar 1991.

[60] S. Park, J. Deyst, and J. P. How, "Performance and lyapunov stability of a nonlinear path-following guidance method," *Journal of Guidance, Control, and Dynamics*, vol. 30, pp. 1718–1728, November-December 2007.