Chapter 1

Introduction

1.1 Navigation

Navigation is a problem as old as autonomous mobile robotics itself. As soon as a robot can move on its own, it must figure out how to move to carry out its objectives or goals. More generally, a robotic or virtual agent will face the problems of motion control and constrained navigation in order to move successfully in an environment. These problems involve finding trajectories through a state-space from one world configuration to another configuration which satisfies some goal criteria. Finding such trajectories is referred to as *motion planning*. Two examples from robotics would be navigation from a robot's current location to some specified goal, or starting from the current robot state and reaching a configuration which is safe with respect to moving obstacles. A critical aspect of motion planning is that it can use simulated actions to evaluate possible trajectories in trying to solve a given problem. It can use this ability to find global solutions which fully solve a problem, whereas an approach using only immediate state information (without action simulation) may get locally "stuck" and fail to find a global solution. The motion planning approach depends on two important properties of the agent and its planner. The first is that the agent has some model of its environment or world state, while the second is that the agent has a model of what its actions do insofar as they affect the world state. Taken together, this allows an agent to predict the result of an action within the environment without actually executing that action, and use this as a primitive to build a motion planner for solving navigational problems.

An important subcategory of autonomous agents are mobile robots, which are free to move around an environment using a locomotive device such as wheels or legs. While an important

ability, this mobility significantly complicates the modelling of the environment and the effects of actions. Most stationary articulated robots can have precomputed models of their working area given to them, while mobile robots normally have to rely on perception to create such a model. Thus mobile robots traditionally were relegated to immediate one-step planning or reactive methods due to the minimal observability offered by sensors. Recent work, particularly with networked sensors and map-building, has allowed mobile robots to gain much better models of their environment, beyond what the sensors can detect at any given time. However, many mobile robots still rely on a reactive layer for local navigation of obstacles, relying only on the current state of the world and mapping it directly to an action. Improved world and action models mean there is an opportunity to use motion planning directly on top of robot control layers. A navigation system employing planning will, however, have to deal with the biggest issues facing mobile robots: The environment can change over time.

1.2 Dynamics

Dynamics is a key issue in navigation for mobile robots in partially structured environments, or more generally for navigation in a multi-agent system lacking explicit coordination. As considered in this thesis, dynamics can be split in agent dynamics and domain dynamics:

- **Agent dynamics** involve the effects of classical physics on the agent itself, resulting in the kinematic and dynamic limitations on the agent which may have to be considered for solving navigation problems. Other remaining differential processes on the agent apply as well, such as electrical and energy storage properties, although the example domains used in this thesis are primarily constrained by classical physics properties. Dynamics constraints limit the acceptable values for derivatives of an agent's position over time, while Kinematic constraints which limit motion along submanifolds of the configuration space. The combined set of constraints are referred to as *kinodynamic* constraints. Kinematic limitations apply at any speed, while dynamics constraints become steadily more important as an agent operates at higher speeds. Robot design cannot escape all agent dynamics issues, as even a holonomic robot lacking any kinematic constraints will face some form of dynamics limitations, and in particular bounds on acceleration and velocity. Thus dynamics limitations are a nearly universal issue for mobile agents.

- **Domain dynamics** involve changes in the problem instance as an agent operates. One cause of such dynamics are changes to the environment. An environment where

an agent operates alone, and obstacles do not change position can be said to be static. An environment where other agents operate, or where obstacles change over time, or even both, can thus be said to be dynamic. Environmental changes can result from classical physics, such as with a moving obstacle acting under known forces. However they can also be driven by other factors, such as discovery of previously unknown obstacles, or updated positions for existing obstacles caused by unmodelled outside influences. Additionally, Domain dynamics can also result from changes in the goal specification over time. This could be due to a higher layer (such as a task-oriented behavior using navigation as a primitive), or because the goal is specified in a dynamic fashion (such as a robot tasked with following another agent).

Another way of classifying overall dynamics into categories is to split based on predictable properties versus unpredictable properties:

- **Predictable dynamics** involve aspects that can be accurately modelled, such as classical physics involving acceleration, velocity, and forces. It can also describe domain changes which evolve in a known fashion over time, such as an obstacle which follows a known trajectory (even if the forces which drive it are unknown.)
- **Unpredictable dynamics** involve aspects that are not modelled, such as alteration of the problem instance by other agents, humans, or exceptional events. Unpredictable dynamics can also describe the gathering of additional information which alters and updates the agent's environment model, even if there was no change to the domain itself. Finally, unmodelled errors in the robot's actions or sensors can also contribute unpredictable dynamics since they alter the problem instance in an unknown way.

A final way of classifying dynamics is by their area of influence:

- **Local dynamics** can be used to describe changes which affect the local area near the agent. An example of local dynamics is the detection of new obstacles nearby a robot using an onboard sensor.
- **Global dynamics** describe changes which affect non-local areas. An example of global dynamics is a robot modifying its world model with map updates sent by other agents.

While the distinction between local and global dynamics may not be particularly interesting in forming a taxonomy or understanding the concept of dynamics, it is important from an

implementational view, as the area which involves dynamics affects the running time of many motion planning algorithms. As a result, the local and global distinction is one way to compare different algorithms when evaluating their suitability for a particular domain.

Now that we have identified several ways with which to classify dynamics, we can describe why such a distinction is useful. Much research has concerned navigation with dynamics, however it has mostly been concerned with predictable dynamics (see Chapter 6 for a detailed description of related work.) As for unpredictable dynamics, it has mainly been addressed in existing research as local unpredictable dynamics, involving changes which occur near the agent. However, a mobile robot may easily face unpredictable global dynamics coupled with predictable local dynamics. Several of the robot platforms considered as targets for this work match that template. Thus, within this thesis we will assume a design goal of operation within a globally unpredictable domain, where changes are made to the environment completely outside of the control of the agent driven by our algorithms. In addition, strong local predictable dynamics are present, affecting the robot's immediate actions. A typical case of such a dynamic workspace is the case of multiple agents operate within the same environment at high speeds.

1.3 Problem Definitions

This section defines the problems of motion planning and cooperative safety as addressed in this thesis. The major notation used for planning and safety are given, along with the definitions of general terms used throughout the document. It concludes with remarks about the complicating factors for motion planning and cooperative safety algorithms.

1.3.1 Motion Planning

In order to form a specification for the motion planning problem as it is applied in this thesis, a more formal definition is adopted. While the planning algorithms described in subsequent chapters are expressed less formally using pseudo-code, each still follows the definition given here. Thus the definition can be thought of as both a specification and a limit on the scope of the problem addressed by this thesis. We now derive a notation similar to Latombe [62], but with some additional generalization.

Definitions:

- W: The robot operates in an environment called a workspace, referred to as W. W has N dimensions (normally $N \in \{2, 3\}$) and is a subset of $\mathbf{R}^N$.
- B: Objects in the workspace that the robot cannot interpenetrate (obstacles) are denoted individually as B_i for some i, and the entire set denoted as B.
- A: A single robot is referred to as A. In the multi-robot case, there are n robots, and each robot is referred to as A_i.
- q: The position of a robot includes values for linear dimensions (such as x and y position), as well as values for orientation (θ) or the angles of various joints. Each of these degrees of freedom (dof) is bounded, and may be modular (such as for orientation). A vector q including all degrees of freedom is called a configuration.
- C: The set of all q values within range defines a subset of $\mathbf{R}^m$ called the configuration space, often abbreviated as *C-space* and denoted as C.
- $C_{free}, C_{contact}, C_{nonfree}$: At each q, the robot A may be penetrating an obstacle, in contact with an obstacle, or not in contact with any obstacle. The set of all configurations where A is inside an obstacle is called $C_{nonfree}$, while the set of obstacle contact configurations without interpenetration is called $C_{contact}$. The configurations where A touches no obstacles is C_{free}. The three sets are disjoint, and their union is C.
- $q_{init}, Qgoal$: The robot starts at position q_{init}, and if an explicit goal position exists it is called q_{goal}.
- G: A goal can more generally be expressed as an evaluation function $G : q \rightarrow \mathbf{R}$. In this case the q_{goal} is set to be the maximum of G over C.
- $\tau(s)$: A path is denoted as $\tau(s)$, and is a continuous function mapping $s \in [0, 1]$ to a configuration in C. A path is constrained by $\tau(0) = q_{init}$.
- *valid*: A path $\tau(s)$ is said to be *valid* if $\forall s.s \in [0, 1] \Rightarrow \tau(s) \in C_{free}$, i.e., the path does not touch any obstacles.
- *solution*: A path $\tau(s)$ is a *solution* if $\tau(0) = q_{goal}$.
- *feasible*: A path may be valid, but due to constraints on the robot's constraints (such as the inability to fly) it may not be executable by the robot. Such a path is *infeasible*, while a path that is within the robot's capabilities it is said to be *feasible*.

The general path planning problem for an explicit goal is:
Given: A, C_{free}, q_{init}, q_{goal};
Find a path $\tau(s)$ which is *valid*, *feasible*, and a *solution*.

We can also distinguish between two variants of the general path planning problem:

Path planning refers to algorithms where s is a pure parameter, and a path-tracking motion controller of some sort will be applied to execute the path.

Motion planning refers to algorithms where s can be mapped to time by some mapping function (i.e. $t = f(s)$). Thus τ defines a trajectory through the state space, based on time. Motion planning is thus a subset of path planning.

Although there are differences between these two types of planning, holonomic robots often blur the distinction with their ability to execute any path at a sufficiently low speed. This thesis deals with both holonomic and non-holonomic robots; The distinction between path and motion planning is mainly reserved for the latter, where the difference is more significant.

1.3.2 Cooperative Safety

In addition to motion planning, we can define a pure version of what it means for a group of robot agents to maintain safety. Note that this is independent of reaching any particular goal, so it is not directly expressible as a kinodynamic [1] motion planning problem with added dynamics dimensions. The definition adopted here is by no means the only way of expressing the concept of safety, and alternate formulations exist (see [32], [46],and [36,37,61]). However, the author is not aware of any alternate formulation that has become dominant, so we adopt our single formulation of cooperative safety for the remainder of this thesis. We have found our formulation to yield a practical algorithms that can be applied to currently existing robots.

We define safety as: Given n agents, where each agent i has a trajectory defined by $q_i(t)$ through state space ($q_i(t) \in C$), and occupies some space $r(q) \in C$ when at position q, then safety at time t is:

[1] *Kinodynamic* refers to combined kinematic (positional) and classical dynamic (position derivatives) properties

$$R_j(t) = \bigcup_{i \in [1,n], i \neq j} r(q_i(t)) \quad (1.1)$$

$$S'(t) = \forall_{i \in [1,n]}\, q_i(t) \in (C_{free} - R_i(t)) \quad (1.2)$$

Where $R_j(t)$ refers to the area covered by all robots *except* j, and $S'(t)$ is the boolean safety function which is true iff all robots are in the remaining free configuration space after removing the areas covered by the other robots. Thus if $S'(t)$ is true, no two robots will collide, and no robot will pass outside of C_{free}. Furthermore, if there are constraints on $\dot{q}(t)$ or $\ddot{q}(t)$ (i.e., velocity and acceleration) in addition to $S'(t)$, we have the problem of *cooperative dynamic safety*. The goal is that agents are able to navigate while maintaining $S'(t)$ at all times, thus ensuring collisions do not take place between agents or with environment obstacles.

1.3.3 Complicating Factors

The difficulty of a particular motion planning or safety problem as we have defined it depends on many parameters, including the complexity of the environment or workspace, constraints on robots' actions, and the ability of the robot to observe and accurately model its environment. The workspace complexity is based on factors such as dimensionality, the number of obstacles, and the complexity of obstacle geometry. Constraints on actions are physical limitations of the robot, such as non-holonomic motion, bounded velocity, or bounded acceleration. Throughout this work, there is a focus on dynamics, which can contribute to motion planning complexity both through predictive difficulty (unpredictable dynamics) and as constraints (predictable dynamics).

1.4 Existing Approaches to Motion Planning

Many design decisions have been made in research on classical navigation problems, and several of those will be adopted in this work The first is the concept of *replanning* to deal with dynamics. The simplest approach is called *unconditional replanning*, where the agent replans each time a new action is to be decided, usually at some regular interval. Alternatively, the agent can plan once, and then monitor the environment and its execution of the plan to determine if it succeeds or fails. If the plan fails during execution, the agent can replan at that time, and then continue its execution. This is called *conditional replanning*. Both

1. Map initial and goal locations to C-space representation
2. Update environment model with new information
3. Update C-space representation graph, or roadmap
4. Search roadmap for a path between initial and goal locations
5. Extract path vertices and edges as plan

Table 1.1: A generic motion planning and replanning algorithm

of this methods alternate planning with execution, and thus can be classified as *interleaved planning and execution*. While some algorithms treat replanning the same as planning from scratch on a new problem instance, other algorithms attempt to carry past information to aid in replanning.

Navigation problems for a mobile agent are often divided into several distinct distance scales, making the overall problem more tractable. Each level feeds a sequence of target locations to lower levels, which lead the robot to the goal. High level planning (such as driving directions on a road network), can largely ignore local motion constraints of the robot, and are global, meaning that a plan is generated all the way from the initial position to the goal position. Mid-level planning refers to methods for navigating a medium size environment (such as within a parking lot), where kinematic constraints on a robot may need to be respected to find a solution (such as parallel parking), but agent dynamics can mostly be ignored. Finally, local planning involves avoiding immediate obstacles while respecting all motion constraints, but is only concerned with reaching goals in a small area, such as out the braking distance. Also, it is common for lower layers to experience more domain dynamics, although this is dependent on the particular environments the agent is operating within. An example of this property is that significant changes in a road network occur less frequently than small obstacles impede the immediate path of a vehicle.

After determining appropriate distance scales and a replanning method, a planning algorithm must be selected. Many options exist, and a detailed overview is provided in Chapter 6, while a short summary will be provided here. First, a generic planning/replanning algorithm is listed in Table 1.1. For a replanning algorithm, the roadmap update step can make use of an existing roadmap from previous queries, while a non-replanning planner will use a new model created from scratch each query. Some algorithms also interleave the roadmap update step (3) with the roadmap search step (4) to gain efficiency for queries covering a small part of the total configuration space.

The planners listed in the related work can be split into two major categories based on the method for generating a roadmap from the configuration space. The two categories are grid-based approaches and randomized sampling approaches. Grid based planners use a regular grid over the environment, with one vertex per cell, and implicit edges connecting each cell to its immediate neighbors. Each cell can have an associated traversal cost, defining it either as free, an obstacle, or assigning some intermediate traversability. If the data source for the environment model is itself a grid, this representation is quite direct, whereas if the data source is not in grid form, a grid must be overlaid on the actual domain, and the cost values of cells populated. The initial and goal locations are mapped to the nearest cells. Edge weights for the implicit roadmap graph are set based on the cell traversal costs, and the resulting graph can be searched using a minimum-cost graph searching algorithm such as Dijkstra's or A^* [72]. The D^* [80] extension of A^* allows for efficient replanning with local updates of the environment around the initial position, as well as updating the initial position itself. D^* updates only the affected portion of the A^* search tree when changes are made to the environment, allowing more efficient update steps when compared to A^*. The popular Field D^* [35] algorithm relaxes the assumption of paths at fixed angles between cells, resulting in shorter paths on average compared to D^*, while sharing its replanning efficiency properties with D^*.

In contrast to grid-based approaches, randomized approaches use random sampling of the C-space in order to construct a roadmap. The Probabilistic Roadmap (PRM) family of planners first samples configurations from C-space at random drawn from some distribution over C-space [53, 54]. First, vertices are drawn at random from the distribution, and the initial and goal configurations are also added as vertices. Then, a local search algorithm attempts to connect nearby pairs of vertices, and if a free path is found, an edge is added between the two vertices representing the local planner's path between the two configurations. This forms a graph of C_{free} which can be searched using A^*. Classical PRM can answer many queries by attempting to connect the initial and goal positions to an existing graph, making it a multi-shot planning algorithm. However, domain dynamics can change the environment, and thus invalidates the roadmap. Some variants of PRM, such as Lazy-PRM, mitigate the construction time by deferring collision checks to the graph search stage, making it more appropriate for dynamic environments. Another family of randomized planners are based on the Rapidly Exploring Random Tree (RRT) concept [64, 65]. This interleaves roadmap construction with search by growing a tree incrementally from the initial position. Configurations are sampled randomly from C-space, and the nearest node in the current roadmap is extended a fixed distance toward the sampled point. The new edge and vertex are added only if the local path is free of obstacles. The sampling and extension steps are repeated to grow a search tree out from the initial position until it reaches a goal state. Many variants exist to improve the search efficiency of RRT, but it is a pure planner without improved efficiency replanning capabilities. The recent DRT [33] planner adds a replanning

mode where environment updates are used to invalidate branches of the RRT tree, and thus it supports efficient replanning for changes near the initial position, such as newly gathered sensor data about obstacles. This is analogous to D^* replanning for grids.

Handling domain dynamics is an important criteria for choosing a motion planning algorithm for a mobile robot, thus it is important to consider the type of dynamics each algorithm can handle. Figure 1.1 conceptualizes the appropriateness of various planning algorithms under conditions of increasing domain dynamics. At the left, there are no dynamics at all (high coherence), while at the right, there are maximum global domain dynamics - each replan is totally unrelated to the one preceding it (no coherence). With little or no dynamics, a static planner without support for replanning is sufficient. In a static domain, the solution need not be updated. Explicit replanners may not be appropriate in this case compared to their static counterparts, as they usually involve some additional overhead (such as with D^* versus A^*). With limited dynamics, such as shifting the initial or goal positions, explicit replanners such as PRM, D^* and DRT gain an advantage, and become the most appropriate planners. Once dynamics changes involve the local environment however, the PRM variants no longer work best, leaving only D^* and DRT. As domain dynamics starts to incorporate global changes however, all of the aforementioned algorithms are no longer appropriate. An example of such a situation would be moving both the initial and goal locations by some amount, as well as modifying the obstacles in the environment without a particular locality of change. At extreme amounts of change however, fast static planners once again become the most appropriate; The change is so great between replans that it is effectively a new planning problem each time. This leaves a large gap between moderate global dynamics and extreme dynamics, into which some mobile robotics applications fall. This thesis introduces the Execution-Extended RRT planner, or ERRT, to address this gap. It makes no assumptions about the locality or type of domain dynamics, and offers a few parameters to tune the planner for the level of dynamics present.

In the case of a multi-agent domain, a navigation system may need to control multiple robots in tandem. Planning for multiple agents can be handled with several different approaches. One conceptually simple method is to concatenate all the agents into a single representation of a joint state space and a joint action space. Unfortunately, the joint space approach leads to poor scalability due to the exponential dimensional scaling of most planning algorithms. This has led to popular decoupled approaches such as *prioritized planning* [87], where the agents plan in a priority order, and later agents avoid the paths planned by earlier agents. While the prioritized planning approach is incomplete, it scales well with the number of robots.

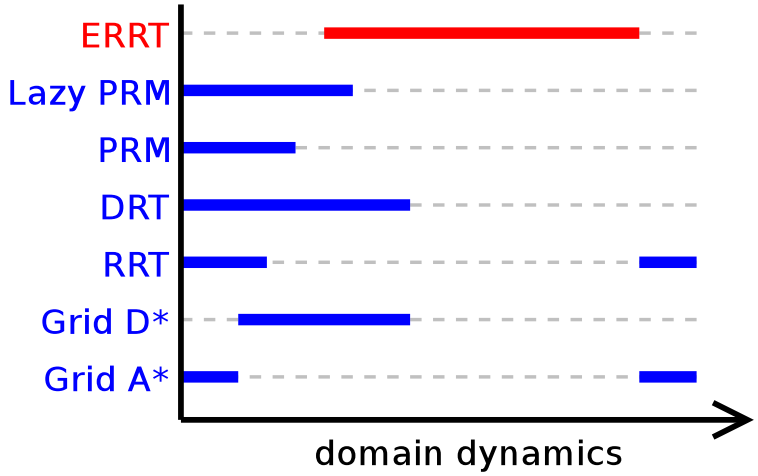

Figure 1.1: A qualitative comparison of the level of domain dynamics targeted by the design of various motion planning algorithms.

1.5 Approach and Thesis

Thesis: Safe multi-robot navigation operating within a real-time constraint is feasible using a combination of randomized motion planning and a cooperative safety algorithm. Furthermore, the approach can yield a practical navigation system for multiple robots operating in unpredictably dynamic environments.

Our approach to motion planning differs from much of the traditional research on motion planning, in that it involves an emphasis on constrained timing requirements as opposed to the solution of increasingly complex environments with calculation time used only as a benchmark. In general, a given problem domain will have some minimum difficulty imposed by the complexity of problem instances, and some maximum time in which a solution must be returned in order for a candidate algorithm to be practical. Such a hypothetical situation is shown in Figure 1.2. An algorithm's performance represents a curve plotting problem difficulty against the calculation time used, which increases with problem difficulty. The time and difficulty constraints can be represented as lines on the plot. The timing constraint imposes a maximum time allowed to solve a problem, thus the algorithm must lie below this line to solve problems of a particular difficulty within the alloted time. The difficulty constraint is a vertical line showing the minimum difficulty that a problem from that domain will entail; While simplifications can be made in problem domains, at some point no further simplification can take place due to the inherent difficulty. An algorithm can be said

to "solve" a problem domain if part of its performance curve lies both below the timing constraint and to the right of the complexity constraint. This is the case for the algorithm shown in Figure 1.2. If one considers the extremes of timing and difficulty constraints, the hypothetical situation shown in Figure 1.3 can arise. In situation (A), the timing constraint dominates, leading to the research question of how difficult a problem can be solved under the timing constraint. In situation (B), the difficulty constraint dominates, leading to the question of how fast a problem of that difficulty can be solved. Note that in each case, a different algorithm is better for each task.

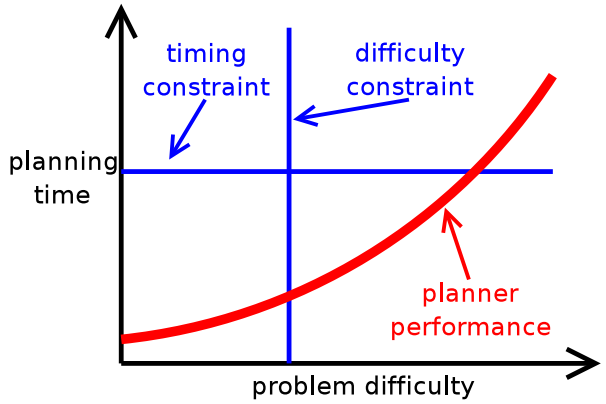

Figure 1.2: A qualitative explanation of planning challenges, plotting problem difficulty against the time used for a hypothetical planning algorithm.

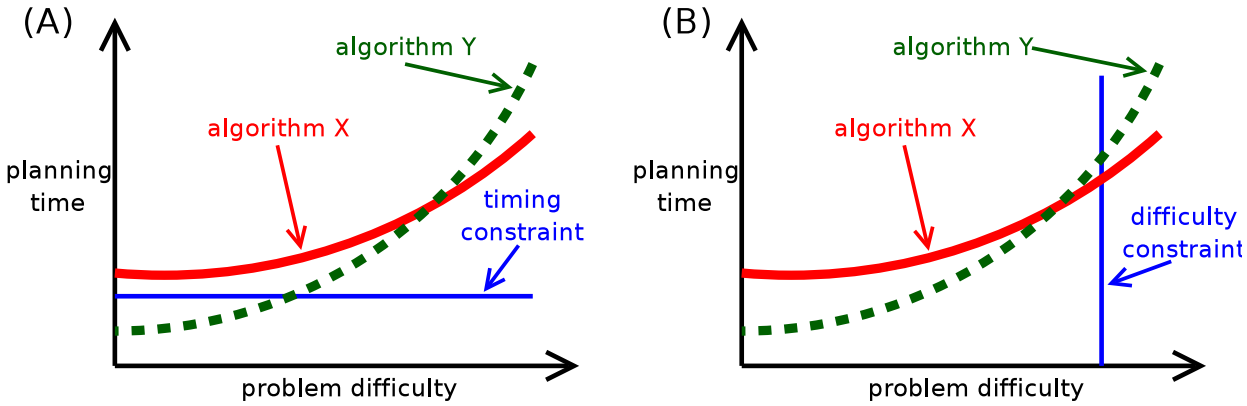

Figure 1.3: A hypothetical comparison of two algorithms under a given timing constraint (A), and a difficulty constraint (B).

This work is very much like the situation shown in Figure 1.3 (A), whereas much of the traditional research in motion planning attacks the problem shown in (B). This is because much

of the traditional work in motion planning is concerned with one-shot queries or multi-shot planning in static environments, and solving such problems in increasingly complex environments characterized by high dimensionality and complex free configuration spaces (See [65] as a typical example). Mobile robots are instead characterized by lower dimensional dynamic environments and nearly continuous re-queries. As a dynamic mobile robot environment can be unpredictable, any path found by a motion planning algorithm has a short period of applicability before it becomes obsolete. This short period leads to an implied iterative control cycle, and imposes a timing constraint on any planner used. In the robot domains considered in this thesis, these timing constraints end up the dominant factor in choosing a planning algorithm. Attempting to solve more complex problems then becomes a dependent factor for comparison, rather than the independent factor as in traditional work.

Thus at its core, this work seeks to extend the boundaries of motion planning, but not in terms of absolute difficulty of problem instances. Rather, it seeks to improve the solution speed of problems, allowing applications where navigation must be carried out within tight timing constraints to solve more complex navigation problems, while remaining within the timing constraint. Additionally, the acceptance of all types of dynamics allows significant changes in the environment and the goal specification can be made between replans, allowing the navigation system to be treated of as a primitive by task-oriented behaviors, rather than treating navigation as an end in itself.

The timing requirements for semi-structured robot domains is dictated by the presence of dynamics, and in particular unpredictable dynamics. A solution must be found and partially executed before the information becomes irrelevant due to unpredicted changes. In robot navigation systems where incomplete or heuristic planners have been used, a common reason for such a choice has been the tight timing constraints imposed by either the domain itself or limited computational resources present on the agent. While some existing mobile robot systems use planning for low-level control, many more do not use motion planning at all, instead relying on local reactive methods for immediate control of the robot, or limited one-step enumeration and evaluation of steering commands. The use of motion planning is restricted to higher levels of a navigation system, or is forgone completely.

The different requirements of mobile robot systems do not mean, however, that recent developments in motion planning for highly complex domains have no place in mobile robotics. What it does mean is that these algorithms must be adapted, and their capabilities improved for these domains to be practical for mobile robot control. Particular progress has been made on traditional planning problems using randomized sampling approaches, such as Probabilistic Roadmaps (PRM) and Rapidly exploring Random Trees (RRT). While the focus of these techniques has been solving difficult problems in cluttered environments and high-dimensional spaces, in this thesis I will demonstrate how they can be adapted for inter-

leaved planning and execution at high control rates in highly dynamic domains. In addition, local dynamics constraints can be handled by a real-time cooperative safety method, complimenting the motion planner's ability to deal with global unpredictable dynamics. The safety method also allows the robots to be planned for individually, ignoring the other robots for the purposes of the path planner, and using the safety algorithm prevent collisions rather than relying on a joint state space or path obstacles from prioritized planning. Specific contributions follow.

1.6 Thesis Contributions

The major contributions of this thesis are the following:

- **The DSS Cooperative Multi-Robot Safety Algorithm:** The novel multi-agent *Dynamics Safety Search* (DSS) algorithm is described. It is based on Fox et al.'s single-agent *Dynamic Window* approach [38]. DSS offers guaranteed safety for single-agent and coordinated multi-agent systems. The *Dynamic Window* approach cannot guarantee exact safety even in the single agent case. DSS is shown to have polynomial complexity (as opposed to exponential complexity for joint planning), and near linear complexity in simulation testing. Finally, DSS is demonstrated to aid in collision avoidance on real robots from the RoboCup small size domain.

- **The Waypoint Cache for Biased Replanning:** The Waypoint cache is a method of using previous plans to alter the sampling distribution for randomized replanners. It allows more efficient replanning in dynamic domains, and is used to develop the ERRT planner.

- **The ERRT Randomized Motion Planner:** The Execution-Extended RRT (ERRT) algorithm is a novel extension of the RRT family of planners. ERRT can be tuned to work for varying levels of domain dynamics faced in a particular application. Bidirectional Multi-Bridge ERRT introduces additional parameters to the Kuffner's RRT-Connect algorithm [51]. These new parameters allow a tradeoff between planner efficiency and plan length optimality.

- **Survey of Collision Detection for Motion Planning:** Existing work on collision detection is evaluated for the task of motion planning, which differs in emphasis from traditional comparisons of collision detection for other applications. The survey of work contributes observations of similarity between several algorithms when applied to motion planning.

- **Navigation for a Competitive Multi-Robot Soccer System:** Navigation lies at the heart of a system with multiple robots working as a team in a soccer domain. In addition to exploring the algorithms individually, robot soccer provides a way to combine path planning and safety methods in the context of a larger system with goals more complex than reaching fixed configurations. Using the algorithm as a primitive for an autonomous soccer system helped drive requirements for robustness and efficiency, while the competition offers a metric to compare implementations from many research groups.

Additional contributions of this thesis are the following:

- The *Extent Masks* collision detection algorithm is a novel contribution of this thesis research. It offers a tradeoff compared to other algorithms, scaling linearly in the number of obstacles rather than sub-linearly, but offers pure linear scaling to any number of dimensions. It it fast in practice for any domain with a small number of obstacles relative to the space occupied or the dimensionality of the environment. (Chapter 3)
- The novel Dynamic PRM algorithm is introduced, which extends Kavraki et al.'s Probabilistic Roadmap (PRM) algorithm [54]. It is tested on the QRIO humanoid robot. (Chapter 4)
- The mathematical approach to iterative swept-sphere collision detection developed in this thesis, although simple, appears to be novel. Quinlan's [75] related approach does not use the explicit iterative formulation. (Chapter 3)
- Requirements for robust motion planning for robotics applications are introduced, and in particular robustness to location error. The author is not aware of these requirements or their offered solutions being identified in existing work on randomized or graph-based motion planners. (Chapter 4)
- The heuristic of an *active goal* is introduced to solve kinematically constrained planning problems where the goal is defined as a fixed-radius orbit around a point. It is demonstrated as part of a planner for fixed-wing unmanned aerial vehicles (UAVs). (Chapter 4)

1.7 Guide to the Thesis

Table 1.7 indicates the chapters of particular relevance to understanding a given contribution.

Contribution	Ch.1	Ch.2	Ch.3	Ch.4	Ch.5	Ch.6	Ap.A	Ap.B
Multi-Robot Safety	•	○	○		★	○	○	
Waypoint Cache	○	○		★				
ERRT Motion Planner	•	○	○	★		•		
Col. Detection Survey		○	★	○				
Robot Soccer System	○	★	•	★	★	○	★	★

Key: ○=Somewhat relevant, •=Relevant, ★=Essential

Table 1.2: Guide to the thesis

Chapter 1 introduces the general problem statements for motion planning and safety and defines the scope of the work.

Chapter 2 describes the robot domains used for motivation and testing of the approaches in this work

Chapter 3 describes the collision detection problem for path planning and several approaches. Very high efficiency is required for practical real-time path planning. A novel approach is introduced for collision detection with sets of obstacles

Chapter 4 describes the motion planning algorithms which extend the existing randomized path planning approaches. Pseudocode and experimental timing data is given. Also describes application extensions.

Chapter 5 introduces the concept of multi-agent cooperative dynamics safety with a novel algorithm for providing it at real-time rates for moderately sized teams.

Chapter 6 explores related work for motion planning, and the most relevant safety method.

Appendix A describes the vision system used to gather positional data for the small-size domain. This chapter supports sensor error models used elsewhere in the document with experimental data

Appendix B explains the software system for the CMDragons small-size team in detail.