Graceful Navigation for Mobile Robots in Dynamic and Uncertain Environments

by

Jong Jin Park

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in The University of Michigan
2016

Doctoral Committee:

Professor Benjamin Kuipers, Co-Chair
Professor Arthur D. Kuo, Co-Chair
Professor Ryan M. Eustice
Professor Jessy W. Grizzle

To my family, and my dear friends.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The ability to navigate in everyday environments is a fundamental and necessary skill for any autonomous mobile agent that is intended to work with human users. The presence of pedestrians and other dynamic objects, however, makes the environment inherently dynamic and uncertain. To navigate in such environments, an agent must reason about the near future and make an optimal decision at each time step so that it can move safely toward the goal. Furthermore, for any application intended to carry passengers, it also must be able to move smoothly and comfortably, and the robot behavior needs to be customizable to match the preference of the individual users. Despite decades of progress in the field of motion planning and control, this remains a difficult challenge with existing methods.

Specifically, we require robot navigation in dynamic and uncertain environments to be safe, comfortable, and customizable. For safety in dynamic and uncertain environments, our algorithm guarantees probabilistic safety, rather than trying to provide absolute collision avoidance which quickly becomes impossible in cluttered and crowded environments. That is, with our algorithm the robot will (i) continually try to minimize (to near zero) the potential cost of collision due to robot motion, (ii) avoid moving when it is already in a collision state, and (iii) try to maximize the progress toward the goal if and only if there exists trajectories that are likely to be feasible. Also, for comfort and customizability, we explicitly consider the quality of motion, but without affecting our probabilistic safety guarantee, so that the robot motion matches the preference of the individual users.

In this dissertation, we show that safe, comfortable, and customizable mobile robot navigation in dynamic and uncertain environments can be achieved via stochastic model predictive control. We view the problem of navigation in dynamic and uncertain environments as a continuous decision making process, where an agent with short-term predictive capability reasons about its situation and makes an informed decision at each time step. The problem of robot navigation in dynamic and uncertain environments is formulated as an on-line, finite-horizon policy and trajectory optimization problem under uncertainty. With our formulation, planning and control becomes fully integrated, which allows direct optimization of the performance measure. Furthermore, with our approach the problem becomes easy to solve, which allows our algorithm to run in real time on a single core of a typical laptop with off-the-shelf optimization packages.

This depends on four specific technical contributions. We define our expected cost so that we can directly incorporate the time-varying uncertain constraints and the probability of violating those constraints into the cost function, which tends to create a smooth cost surface that is easy to optimize over. The dimensionality reduction of this problem critically depends on the policy and closed-loop trajectory parameterization based on a Lyapunov-based feedback control law, which we developed for graceful motion of differential wheeled mobile robots. The stability of this stochastic model predictive control critically depends on a non-holonomic distance function, which we define as a Control-Lyapunov function for unicycle-type vehicles. We also develop a motor and friction dynamics model of the robot for more accurate forward prediction.

We demonstrate that our method generates graceful (safe, smooth, comfortable, fast, and intuitive) and customizable robot behavior in physical environments with pedestrians in real time. The work presented in this thesis extends the state-of-the-art in analytic control of mobile robots, sampling-based optimal path planning, and stochastic model predictive control. We believe that this work is a significant step toward safe and reliable autonomous navigation that is acceptable to human users.

CHAPTER I

Introduction

This dissertation shows that safe, comfortable, and customizable mobile robot navigation in dynamic and uncertain environments can be achieved via stochastic model predictive control. We formulate local navigation in dynamic environments as an on-line, finite-horizon policy and trajectory optimization problem under uncertainty, which becomes easy to solve with our approach.

1.1 Background and Motivation

Moving robots are fascinating. It is fun to watch robots move, especially when the robot's motion is agile and visibly intelligent. Moreover, mobile robots capable of autonomous navigation have the potential to greatly benefit human users, by assisting mobility and making everyday life safer and more comfortable. This has increased public attention recently, because of the series of successful DARPA Challenges and wide news coverage of autonomous car projects from Google and virtually all major car manufacturers.

Indeed, planning and control for mobile robots has been a very active area of research for the past few decades with a large body of existing research. As a result, we already have mobile robots in our homes (robot vacuums), on factory floors (Kiva Systems), and some simple autonomy in our cars under controlled situations like adaptive cruise control and parallel parking. Yet still, safe, comfortable, and customizable mobile robot navigation in fully dynamic and uncertain environments is generally regarded as a very difficult problem.

1.1.1 Traditional Approaches to Motion Planning and Control

Why is it so hard?

The key difference between factory floors and everyday environments is the presence of other autonomous agents (e.g. pedestrians). The presence of other agent make the environment inherently dynamic and uncertain, which requires the motion planner to have much higher flexibility and a direct way to handle uncertainties without losing the stability guarantee and precision in control. Also, for passenger-carrying vehicles, comfort and customizability are also essential.

Traditionally, a typical motion planning and control architecture consists of four distinct and independent processes (Fig. 1.1, Left): (1) geometric path finding, which finds a collision-free path given a task and a (known) map; (2) path smoothing, which modifies the found path to satisfy differential constraints; (3) trajectory planning, which computes a velocity profile that tracks the path; and (4) a feedback controller that receives the desired velocity as a control target and generates low-level commands that allows the robot to track the trajectory, which is the place where robot dynamics and feedback first enters the picture. See *LaValle* (2011a,b) for a brief overview; for more comprehensive reviews *Choset et al.* (2005) and *LaValle* (2006) provide excellent texts. We also give an overview of existing algorithms in Chapter II.

This classical structure is simple and straightforward. In situations where flexibility and uncertainty handling do not matter as much, the classical approach can work very well. After all, the basic problem of computing a collision-free path for a robot among known obstacles is well understood and reasonably well solved (*LaValle*, 2011b).

This classical structure (Fig. 1.1, Left), however, fundamentally limits the flexibility of the system, as each process is completely confined within a very small search space defined by the output of the previous one. It is also not clear how predicted future motion and associated uncertainties of other dynamic objects can be handled in this framework. In addition, as *Gulati* (2011) pointed out, it is difficult to optimize a performance measure in the final trajectory with this framework: Since the individual processes are decoupled to each other in this structure, optimization in each step does not lead to optimality in the overall performance. Safe, comfortable, and customizable mobile robot navigation in fully dynamic and uncertain environments requires a method that is more flexible, can handle uncertainties in future predictions, and can optimize robot behavior to the preference of its user.

1.1.2 Robot Navigation via Probabilistic On-line Decision Making

How do we solve this problem? Let us first consider how people drive — or at least how people are taught to drive — on roads. Driving is a continuous decision making process. At each time step, a driver needs to (i) identify the current situation, (ii) predict how things

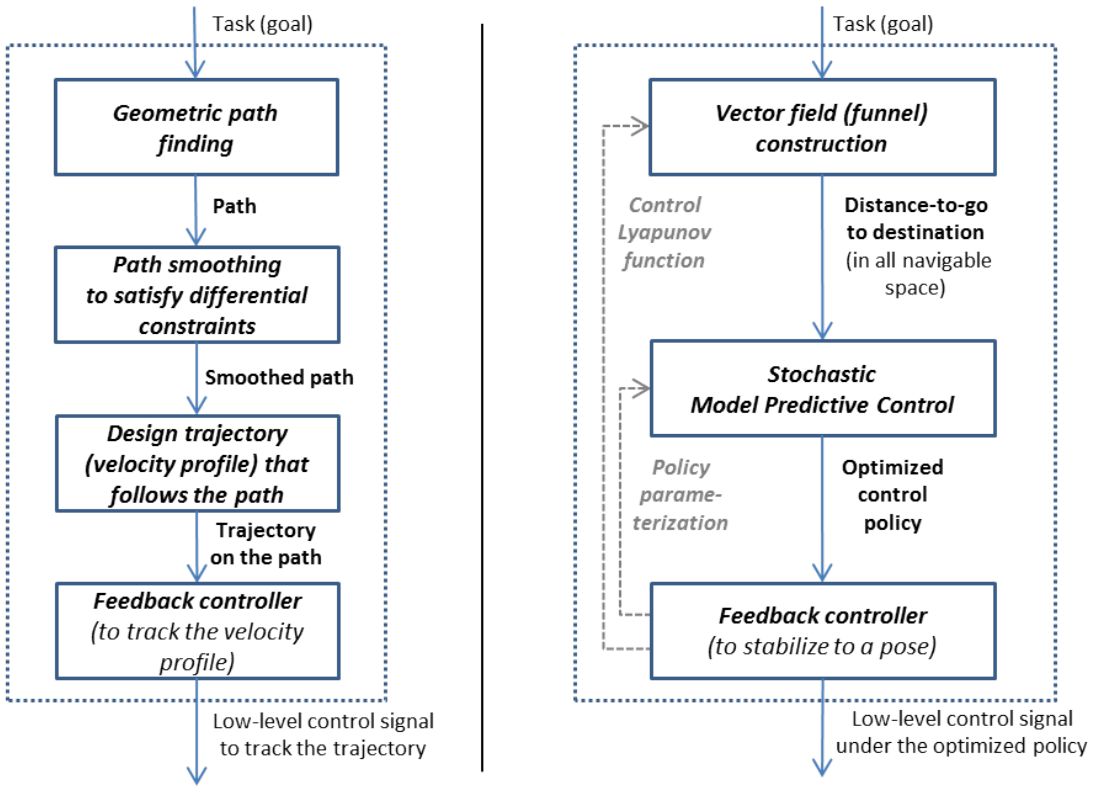

Task (goal)
Geometric path finding
Path
Path smoothing to satisfy differential constraints
Smoothed path
Design trajectory (velocity profile) that follows the path
Trajectory on the path
Feedback controller (to track the velocity profile)
Low-level control signal to track the trajectory
Task (goal)
Vector field (funnel) construction
Control Lyapunov function
Distance-to-go to destination (in all navigable space)
Stochastic Model Predictive Control
Policy parame-terization
Optimized control policy
Feedback controller (to stabilize to a pose)
Low-level control signal under the optimized policy

Figure 1.1: Motion planning and control architectures: Traditional vs. Ours.

Left: A typical motion planning and control architecture, which consists of path finding, path smoothing, trajectory design, and feedback control. Planning and control are decoupled, and each process is confined within a very small search space defined by the output of the previous process. Complete knowledge of the robot and other obstacles are often assumed, and sensor feedback is utilized only at the final stage.
Right: Our approach, where planning and control are tightly integrated. We first compute the distance-to-go to the destination in all navigable space, where the distance is defined by the control Lyapunov function of the underlying controller. Then we have our stochastic model predictive control which directly optimizes the local control policy for the feedback control, using the computed distance-to-go, the estimated future states of dynamic objects, and the associated uncertainties. The overall system is much more flexible and can handle the variability and the uncertainties of real environments.

will change in the near future, (iii) decide what to do based on the prediction, then (iv) execute the decided action. This process of *Identify, Predict, Decide, and Execute* (IPDE) forms the basis of safe driving which every driver should practice.

We want our motion planning algorithm to do the same, so that the robot can properly handle the uncertainty and variability of real environments and navigate safely. In other words, we view the problem of navigation as a *probabilistic on-line decision making process* where an agent with short-term predictive capability reasons about its situation and makes an informed decision at each time step. An overview of our approach which implements this view is described in Section 1.2.

1.2 Overview of Our Approach

We formulate local navigation in dynamic and uncertain environments as an on-line, finite-horizon policy and trajectory optimization problem under uncertainty, so that the trade-offs between progress toward the goal, quality of motion, and possible collisions are fully considered. The overall architecture is described and compared to the traditional approach in Fig. 1.1.

First, we compute the distance-to-go to the destination at all points in the navigable space. This distance measure is defined by the control Lyapunov function (CLF) (*Khalil*, 2002) of the underlying controller, so that this measure naturally reflects the non-holonomic, differential constraints of the system. We also show that the popular Euclidean metric is not a proper choice for systems with differential constraints (*Park and Kuipers*, 2015). This computation of the distance-to-go over all navigable space can be done efficiently using navigation function based approaches, for example (*Konolige*, 2000), or our non-holonomic RRT* (*Park and Kuipers*, 2015).

Second, we have a stochastic model predictive controller that optimizes the local control policy using the computed distance-to-go. Note that our method is much more flexible compared to a typical approach where only a controller follows a pre-determined path. Our stochastic model predictive controller was first introduced as model predictive equilibrium point control (MPEPC) (*Park et al.*, 2012a,b; *Park and Kuipers*, 2013), where we have shown that the policy optimization problem can be solved efficiently using (i) a compact parametrization of the trajectory by a pose-stabilizing feedback control law (*Park and Kuipers*, 2011), and (ii) an adaptive cost formulation that incorporates the probability of constraint violation (e.g., collision) directly into the (expected) cost definition. The algorithm provides real-time control of a physical robot, while achieving comfortable motion, static goal reaching, and dynamic obstacle avoidance in a unified framework.

Finally, a pose-stabilizing feedback controller (*Park and Kuipers*, 2011) is responsible for steering the robot to a local target with fast feedback and noise rejection.

Our system is designed to handle the uncertainty and variability of real environments. The stochastic MPC and the pose-stabilizing feedback control grant much higher flexibility to the system compared to traditional path following and associated velocity tracking. Also, planning and control are tightly integrated in our approach, which allows direct optimization of any performance measure in the final trajectory.

The overall architecture is constructed hierarchically in a time scale. The first step is finding the distance-to-go, which replaces path planning and smoothing. It is responsible for finding an approximate solution over a large time horizon (a few minutes or more) using the kinematics model. Then, using this distance-to-go as a guide, stochastic model predictive control is responsible for finding a time-optimal solution within a finite time horizon (a few seconds) using a higher-order dynamics model. Here, the control policy of the underlying feedback control is directly optimized to ensure that the resulting trajectory is safe, smooth, and comfortable, and that the robot makes progress toward the goal. Finally, the pose-stabilizing feedback is responsible for noise rejection at a very small time scale (tens of milliseconds).

1.3 Contributions

For applications that are intended to carry human passengers, the problem of navigation requires not only a trajectory from start to goal, but one of high quality: It must be safe, smooth, fast, and intuitive in the presence of dynamic and uncertain hazards to be avoided.

In this dissertation, we show that safe, comfortable, and customizable mobile robot navigation in dynamic and uncertain environments can be achieved via stochastic model predictive control. We view the problem of navigation in dynamic and uncertain environments as a continuous decision making process where an agent with short-term predictive capability reasons about its situation and makes an informed decision at each time step. The problem of robot navigation in dynamic and uncertain environments is formulated an on-line, finite-horizon policy and trajectory optimization problem under uncertainty. With our formulation, planning and control become fully integrated, which allows direct optimization of performance measures. Furthermore, we show that, with our approach, the problem becomes easy to solve, which allows our algorithm to run in real time on a single core of a typical laptop with off-the-shelf optimization packages.

Specific contributions of this work include, for each Chapter,

- *Development of a smooth control law for unicycle-type vehicles in 2D (Chapter III)*

  We have formulated the kinematic control law (3.13) and the pose-following algorithm for smooth and comfortable motion of unicycle-type robots. The control law steers the vehicle to an arbitrary pose in space in a smooth and intuitive curve. The characteristic of robot motion can be easily manipulated via choice of two parameters k_ϕ (orientation weight) and k_δ (gain on heading correction). We also provide a path following strategy that allows the vehicle to traverse the environment satisfying user-imposed bounds in higher derivatives of velocities, given sparsely placed target poses as path instructions. The proposed control law and the pose following strategy is important since it provides safety and comfort to the user without sacrificing maneuverability, and also avoids actuator overload and makes the path physically realizable.

- *Non-holonomic distance function and optimal path planning for unicycle-type vehicles (Chapter IV)*

  We present a non-holonomic distance function for unicycle-type vehicles, and use this distance function to extend the optimal path planner RRT* to handle non-holonomic constraints for unicycle-type vehicles. The critical feature of our proposed distance function is that it is also a control-Lyapunov function, so it better represents the true cost-to-go between configurations and properly reflects the constraints of the system. It allows us to readily generate smooth, intuitive, and feasible paths. By using provably stable control laws and the closed-form distance function that properly reflects the constraints, our algorithm finds smooth and precise paths that exactly reach their goals for unicycle-type vehicles, and provides the stabilizing vector field and the cost-to-go to the final destination around the planned path by composition of local control-Lyapunov functions.

- *Policy and trajectory optimization via stochastic model predictive control for mobile robot navigation in dynamic and uncertain environments (Chapter V-VI)*

  We define our *expected cost* so that we can directly incorporate the time-varying uncertain constraints, and the probability of violating those constraints, into the cost function. The probability is used to generate a time-varying weight that automatically balances the progress toward the goal, the cost of action, and the cost of constraint violation (collision) to compute the overall expected cost. This allows the robot to exhibit reasonable and seemingly intelligent behavior across a wide range of real and challenging situations. We then provide a compact parameterization of the infinitedimensional trajectory space that makes the optimization on-line tractable, which critically depends on the policy and closed-loop trajectory parameterization based on our Lyapunov-based feedback control law from Chapter III. Also, we note that the stability of this stochastic model predictive control critically depends on the use of our non-holonomic distance function from Chapter IV (which is also a control-Lyapunov function for unicycle-type vehicles) as the terminal cost of the MPC.

- *High-fidelity Model of Robot Dynamics with Friction (Appendix)*

  Having an accurate model for robot dynamics is an essential prerequisite for successful implementation of an MPC. We present a straightforward dynamics model of our driving platform (a commercially available powered wheelchair retro-fitted with lidar sensors, an inertial measurement unit (IMU), and odometers), incorporating simple non-linear friction and motor saturation. We show that our physics-based model can accurately represent the dynamics of the real system.

The work presented in this thesis extends the state-of-the-art in analytic control of mobile robots, sampling-based optimal path planning, and stochastic model predictive control. We believe that our work is a significant step forward toward safe and reliable autonomous navigation that is acceptable to human users.

CHAPTER II

Related Work

There exists a large body of work in motion planning. In this chapter, we give a brief overview on classical methods, and then review more recent advances in the field.

The space of all possible positions and orientations of a robot is called the *configuration space*. A series (an ordered set) of points in the configuration space of the robot is called a *path*, and a time-parameterized series of points in the space is called a *trajectory*.

2.1 Overview on Classical Methods

2.1.1 Geometric Path Planning

Geometric path planning is the process of finding a collision-free path in the environment, where usually the complete (geometric) knowledge of the environment is assumed. In classical approaches, the dynamics of the robot and the environment are often ignored, and the path found by these methods requires a separate controller that realizes this path in the physical system. See Section 3.1 for further review on this topic.

Dijkstra's Algorithm (*Dijkstra*, 1959) and the A^* (*Hart et al.*, 1968) are well-established algorithms that can find a shortest path on a graph and on grid-based maps. These algorithms are straightforward best-first search, which can quickly recover optimal solutions in wide range of configurations.[1] Incorporating robot dynamics, high-dimensional space, non-circular robot shape, and dynamically changing environments, however, are serious challenges along these lines of work, and there are a vast number of publications which address those issues. It is interesting to note that the majority of participants used the A^* and its variants (*Ferguson and Stentz*, 2007; *Dolgov et al.*, 2010) to find an optimal path, or at least an initial guess of the path, due to its ability to quickly find a solution under those simplifying assumptions.

[1]See any planning textbooks, e.g. *LaValle* (2006); *Choset et al.* (2005); *Thrun et al.* (2005) for details.

Another very powerful and well-established family of planning algorithms rely on sampling. These sampling-based methods, such as rapidly-exploring random tree (RRT), probabilistic roadmap (PRM), and their variants (*LaValle*, 1998; *LaValle and Kuffner*, 2000), can work in a high-dimensional continuous space and can incorporate differential constraints and robot shape naturally from problem formulation with a guarantee to find a feasible path if it exists, given sufficient time. These algorithms try to construct a topological graph that connects the start and the goal, where an edge represent a traversable path segment and a node is a point in free space.[2] The major problem with the original formulation was optimality. In fact, in *Karaman and Frazzoli* (2011), where the asymptotically optimal sampling-based motion planner (RRT*) is introduced, it is shown that the classical sampling-based planners can never find the optimal solution. This is discussed further in Section 2.2.

2.1.2 Reactive Methods

The paths found by geometric path planning algorithms typically require non-trivial and often expensive post-processing to make the path smooth and admissible to the controller. In dynamic environments, the plan often needs to be recomputed entirely when dynamic obstacles block the path or when the target moves.

In highly dynamic environments, local and immediate motions become more important to avoid collisions, so reactive, control-oriented methods are often more desirable. Some early work includes potential field methods (*Khatib*, 1986; *Koren and Borenstein*, 1991; *Rimon and Koditschek*, 1992) and vector field histogram (VFH, *Borenstein and Koren* (1991)), where a robot is reactively pushed away from obstacles and pulled toward the goal. These methods were originally introduced as reactive obstacle avoidance (*Khatib*, 1986) and later extended to path planning and navigation (*Rimon and Koditschek*, 1992).

A potential function can suffer from local minima, causing the robot navigating using the potential field to get stuck. There are navigation function based methods (see *LaValle* (2006)) and gradient methods (*Konolige*, 2000; *Konolige et al.*, 2008) which can eliminate local minima associated with naive potential fields, where the navigation function is constructed so that it has only a single minimum at the goal and has a monotonic gradient. It is difficult, however, to consider complex robot and obstacle shapes, and it quickly becomes expensive to compute the field in high-dimensional space.

Dynamic window approach (DWA) (*Fox et al.*, 1997), and velocity obstacle (VO) and their variants (*Fiorini and Shiller*, 1998; *van den Berg et al.*, 2008) are methods that search

[2]See any planning textbooks, e.g. *LaValle* (2006); *Choset et al.* (2005); *Thrun et al.* (2005) for details.

the velocity space directly and chooses a velocity that is guaranteed to avoid collision. Velocity obstacle is defined as the set of all velocities of a robot that will eventually lead to collision. These methods are interesting because of their dynamic replanning framework, and because their formulation naturally allows incorporation of interaction between agents in the environment (e.g. *van den Berg et al.* (2008); *Snape et al.* (2011)). These methods, however, are formulated at a kinematic level, and assumes accurate knowledge of the environment and other agents. Thus, in general, it is difficult to incorporate higher-order dynamics and uncertainties into these framework.

2.2 Recent Advances

2.2.1 Dynamic Replanning

Many of the finalists in the 2007 DARPA Urban Challenge used dynamic replanning framework, where the robot actively choose an action that is expected to maximize the benefit in the near future. The action is often chosen from a pre-defined set of feasible paths or control sequences, which are often called *motion primitives* (*LaValle*, 2006). This includes constant-curvature arcs and pre-defined path sets (*Ferguson et al.*, 2008; *Von Hundelshausen et al.*, 2008; *Rauskolb et al.*, 2008; *Bohren et al.*, 2008; *Bacha et al.*, 2008). Although full direct treatment of uncertainties and the customizability of motion are lacking, the success of these algorithms demonstrate the utility of the dynamic replanning framework.[3]

The concept of dynamic replanning with model-based prediction is well-established in the control community as model predictive control (MPC). *Mayne et al.* (2000), *Rawlings* (2000), *Lee* (2011) are excellent theoretical reviews. There are many applications in robot systems, e.g. *Singh and Fuller* (2001), *Frew* (2005), *Bertrand et al.* (2006) for UAVs and *Schouwenaars et al.* (2004), *Ogren and Leonard* (2005), *Howard et al.* (2009), *Droge and Egerstedt* (2011) for ground vehicles. MPC is useful in dynamic and uncertain environments as it provides a form of information feedback with constant re-planning by continuously incorporating new information with the receding time horizon and optimizing the prediction of robot behavior within the time horizon. In the optimization framework, trade-offs between multiple objectives can be expressed explicitly in the objective function; however, with existing methods it is not clear how to handle uncertainties. See Section 5.1 for further discussion on this topic.

[3]Also see *Knepper and Mason* (2009, 2012).

2.2.2 RRT*

RRT* (*Karaman and Frazzoli*, 2011) is the current state-of-the-art in sampling-based motion planning. It is an extension of the classic RRT algorithm with added functionality to rewire and streamline existing graph structure. In *Karaman and Frazzoli* (2011), it was proven that the cost of the best path returned by RRT converges almost surely to a non-optimal value as the number of samples increases, and the RRT*, a sampling-based planner with asymptotic optimality guarantee, was proposed. *Karaman et al.* (2011) provides a nice implementation example, and *Karaman and Frazzoli* (2013) discusses the extension of the algorithm to systems with non-holonomic constraints. See Section 4.1 for further discussion on this topic.

2.2.3 Comfortable and User Customizable Motion

Only limited attention has been given to planning and control for comfortable and user-customizable motion, partly due to the difficulty of quantifying comfort/discomfort in the first place. *Gulati* (2011) provides the most comprehensive results to date, solving a full optimization problem over the entire trajectory space to minimize a carefully designed discomfort metric. Uncertainties regarding dynamic objects and non-circular robot geometry are not considered in this example. Other works (*Indiveri et al.*, 2007; *Gulati et al.*, 2009) are typically modifications of existing control laws to render the trajectory more comfortable. Another notable work is *Nagarajan et al.* (2013), where an integrated motion planning and control algorithm for graceful motion of balancing robot is proposed. This method achieves good results in a relatively open area, but this method depends on pre-allocation of a finite number of motion policies to free-space, which does not work in tightly constrained environments.

2.2.4 Planning in Dynamic and Uncertain Environment

Planning in dynamic and uncertain environments is an active area of research, and important results have been published recently (*Du Toit and Burdick*, 2012; *Aoude et al.*, 2013; *Trautman et al.*, 2015; *Blackmore et al.*, 2006; *Lambert et al.*, 2008; *Luders et al.*, 2010; *Du Toit and Burdick*, 2011; *Trautman et al.*, 2013).

Du Toit and Burdick (2012) presents a formal treatment on planning under uncertainties, where the predicted motion uncertainties of the robot and other agents are evaluated as a stochastic dynamic program. Intractability is avoided with the receding horizon technique, and the authors show that the navigation performance is improved by assuming the most likely measurements which makes the robot less conservative in uncertain situations. The method is evaluated for disk robots with disk objects in simulations. For safe navigation, they impose a hard threshold on acceptable probability of collision. (This is the so-called chance constraint (*Charnes and Cooper*, 1959; *Luders et al.*, 2010; *Aoude et al.*, 2013)).

In *Aoude et al.* (2013), sophisticated models of motion patterns of dynamic objects in the environment are learned via the Gaussian Process (GP) mixture model, based on observed past trajectories of dynamic obstacles. Future trajectories of dynamic objects are estimated via the learned GP and RRT-based simulations to check for collision. The presented robot navigation algorithm, called chance-constrained RRT, is also based on the RRT algorithm with the constraint on minimum acceptable probability of collision (the chance constraint). The method is evaluated in simple simulated environments, and optimality (and hence the quality of motion) is not considered.

Trautman et al. (2015) focuses on modeling interactions between the robot and other agents. They convincingly argue that if the robot cannot anticipate human cooperation (i.e. pedestrians will also try to avoid the robot), then navigation in crowded environments is impossible. A GP-based pedestrian model is developed (interacting GP) where the interaction and intention of other agents are modeled. The method is tested in a relatively small cafeteria environment (6m travel) with a slow-moving (0.3m/s) round robot based on the rate successful traversal. Quality of individual trajectories were not evaluated.

CHAPTER III

A Smooth Control Law for Graceful Motion

[1] Although recent progress in 2D mobile robot navigation has been significant, the great majority of existing work focuses only on ensuring that the robot reaches its goal. But to make autonomous navigation truly successful, the 'quality' of planned motion is important as well. Here, we develop and analyze a pose-following kinematic control law applicable to unicycle-type robots, such that the robot can generate intuitive, fast, smooth, and comfortable trajectories.

The Lyapunov-based feedback control law is derived via singular perturbation. It is made up of three components: (i) egocentric polar coordinates with respect to an observer on the vehicle, (ii) a slow subsystem which describes the position of the vehicle, where the reference heading is obtained via state feedback, and (iii) a fast subsystem which describes the steering of the vehicle, where the vehicle heading is exponentially stabilized to the obtained reference heading. The resulting path is a smooth and intuitive curve, globally converging to an arbitrary target pose without singularities, from any given initial pose.

Furthermore, we present a simple path following strategy based on the proposed control law to satisfy arbitrary velocity, acceleration and jerk bounds imposed by the user. Such requirements are important to any autonomous vehicle so as to avoid actuator overload and to make the path physically realizable, and they are critical for applications like autonomous wheelchairs where passengers can be physically fragile.

3.1 Planar Motion Control

Planar motion control is a fundamental problem for any autonomous mobile platform, and it has been a very active area of research for the past few decades with a large body of existing literature. Many existing approaches for planar motion control involves tar-

[1]This chapter is a revised (and expanded) presentation of *Park and Kuipers* (2011).

get tracking, with targets being positional waypoints or attached to a predefined pathway (*Breivik and Fossen*, 2008).

Waypoint-following control is common in the fields of aerospace and naval sciences, where orientation of a vehicle arriving at a desired target carry less importance as in the case of missile guidance or control of surface vessels (*Bakaric et al.*, 2004; *Aguiar and Pascoal*, 2007). The method is simple and intuitive, and it can be robust to disturbances since there is no associated deviation-from-path error. But with this method, smooth transitions between waypoints is an inherent problem; the orientation and velocity at a given location significantly influence subsequent motion and control effort, but positional waypoints do not provide constraints on those quantities.

Path-following control addresses this issue by first designing the entire pathway and then making the system converge to the path (*Micaelli and Samson*, 1993; *Lapierre et al.*, 2006; *Piazzi and Bianco*, 2004; *Magid et al.*, 2006). With control over the design of the pathway, tasks which can be difficult to solve with waypoint-following such as static obstacle avoidance can readily be solved. However, this approach does require accurate and explicit path planning, and in general it does not work well with dynamic obstacles like pedestrians. Also, with current methods it is not clear how to make motion along the path smooth and comfortable, i.e. with bounded velocity, acceleration, and jerk. The situation can be more problematic when the path is designed independently of vehicle dynamics, which is typical of path design by parametric curves such as B-splines.

In terms of comfortable motion, other popular classical methods such as vector field histogram (VFH) (*Borenstein and Koren*, 1991) or dynamic window approach DWA (*Fox et al.*, 1997) are also not suitable (*Gulati and Kuipers*, 2008). Since these methods do not impose constraint or penalty on acceleration and jerk, the optimal control selected by these methods can be arbitrarily large. Likewise, the great majority of existing control methods assumes ideal actuators, i.e. arbitrarily large velocity or torque can be generated instantly (*Indiveri et al.*, 2007). Such an assumption fails to sufficiently account for the robot dynamics, which in turn can lead to failure in navigation (*Konolige et al.*, 2008).

The issue becomes even more critical in the case of an autonomous wheelchair, which is the primary target application of the control methods proposed in this paper. An intelligent wheelchair which is capable of autonomous and safe navigation between distinct points in an environment can provide a necessary level of motor assistance to individuals suffering from physical or mental disabilities. But control methods which do not limit the velocity, acceleration and jerk are clearly not suitable for wheelchairs since high acceleration can harm passengers even without a collision, in addition to being uncomfortable. Furthermore, we also stress that the path should be intuitive (appear natural) enough to be acceptable to human passengers. Note that these are general requirements to all autonomous vehicles intended to carry human passengers, but the issues are more pronounced for wheelchairs.

Thus we require the motion of the robot to be *graceful*. We use the qualitative term *graceful* as introduced by *Gulati and Kuipers* (2008) to refer to a motion visibly safe, comfortable, fast, and intuitive. To be safe and comfortable, a motion needs to be smooth with bounded velocity, acceleration and jerk.

Only limited attention has been paid to this subject. In *Indiveri et al.* (2007), a path following control law is modified to limit actuator velocity. The approach in *Gulati and Kuipers* (2008) also builds upon an existing path following control law (*Lapierre et al.*, 2006) and achieves bounds in angular velocity and angular acceleration by adjusting linear velocity according to path curvature. In *Gulati et al.* (2009), the trajectory and associated control signal is fully generated via numerical optimization with a cost function which provides graceful motion, but without a closed-form control law.

In this paper, we take a different approach to achieve graceful motion. We first show that with a natural choice of a coordinate system, a simple and robust kinematic control law can be found via singular perturbation (*Khalil*, 2002), which guides a wheelchair from a given pose (position and orientation), guaranteeing convergence to an arbitrary motion target (a prescribed position, orientation, and linear velocity), following a smooth, intuitive curve. Then we show that this closed-form control law makes it possible to design a set of motion targets to guide the robot toward a specified goal with bounded velocity, acceleration and jerk.

3.2 Egocentric Polar Coordinates

It is well known that differential drive cart and simple cars can be modeled as a simple unicycle *LaValle* (2006). Consider a vehicle described by the unicycle model implemented with two independently driven parallel wheels such that linear velocity and angular velocity can be controlled independently. The vehicle is underactuated in the sense that the linear velocity of the vehicle is always aligned with the orientation of the vehicle (i.e. the vehicle cannot move sideways).

We adopt a polar coordinate system to describe the vehicle kinematics. With this coordinate system, we try to identify and follow how human drivers observe and describe a target, such that the derived control law using the coordinate system can produce motion which appears natural to human passengers.

Suppose an observer, or a sensor, is situated on a vehicle and fixating at a target p_0 at a distance r away from the vehicle. Let $\phi \in (-\pi, \pi]$ be the orientation of p_0 with respect to

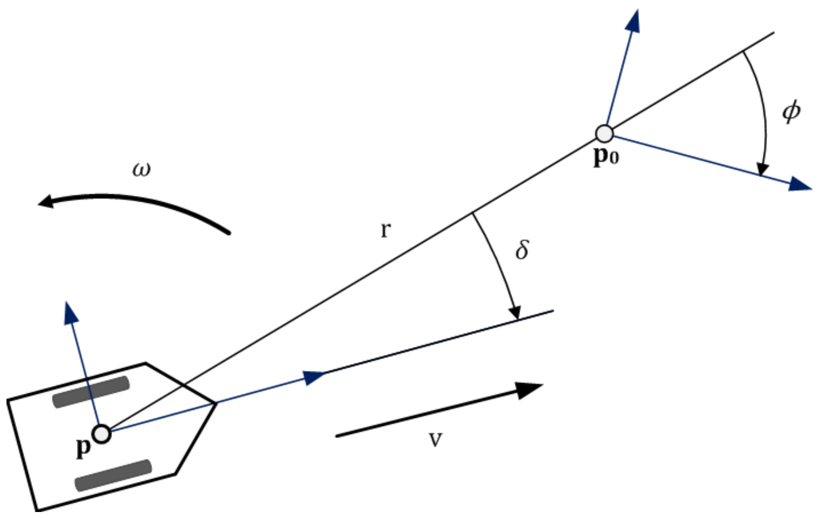


Figure 3.1: Egocentric polar coordinates

From an observer situated on a vehicle at pose p fixating at target pose p_0, r is the radial distance to the target, ϕ is the orientation of p_0 with respect to the line of sight from the observer to the target, and δ is the orientation of the vehicle heading with respect to the line of sight. Here, both ϕ and δ have negative values. At $r = 0$, we take the line of sight to be aligned with the target orientation and set $\phi = 0$, and δ is the the vehicle orientation measured from the target orientation.

the line of sight from the observer to the target. And let $\delta \in (-\pi, \pi]$ be the orientation of the vehicle heading with respect to the line of sight (LOS), as shown in Fig. 3.1.

Then, it is easy to show that the vehicle kinematics can be written as

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} -v \cos \delta \\ \frac{v}{r} \sin \delta \\ \frac{v}{r} \sin \delta + \omega \end{pmatrix} \tag{3.1}$$

where v and ω are linear and angular velocity of the vehicle, respectively. Observe that $\dot{r} = -v \cos \delta$ and $\dot{\phi} = \frac{v}{r} \sin \delta$ are the usual equations of motion of a point particle in polar coordinates where ω plays no role. The change in vehicle heading $\dot{\delta}$ is influenced by the change in the direction of the LOS due to vehicle movement as well as ω, such that $\dot{\delta} = \frac{v}{r} \sin \delta + \omega$. Observe that if $\omega = 0$, then $\dot{\delta} = \dot{\phi}$. A nearly identical set of state equations can be found in *Aicardi et al.* (1995), but wherein the coordinates are defined with respect to the target rather than the observer.

3.3 Pose-stabilizing Control Law

From the previous section, we have $(r, \phi, \delta)^T$ as our coordinates of error space where the control problem is developed and solved. The control problem of moving a vehicle from any given initial pose to a target pose T becomes the problem of bringing $(r, \phi, \delta)^T$ to the origin. We treat velocity commands v and ω as the control variables, i.e. the controller is developed at kinematic control level.

To begin, let us assume v is some nonzero positive (not necessarily constant) and ω is the only control signal. Then from (3.1), it can be seen that the control signal ω only affects the state δ, and $(r, \phi)^T$ are determined via δ. Also observe that r and ϕ completely describes the position of the vehicle, and δ corresponds to steering the vehicle. Thus we can decompose the system (3.1) into two parts as follows.

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\delta \\ \frac{v}{r}\sin\delta \end{pmatrix} \tag{3.2}$$

$$\dot{\delta} = \frac{v}{r}\sin\delta + \omega \tag{3.3}$$

This decomposed structure of the system motivates a control strategy via singular perturbation, or two-time scale decomposition. The idea is to find (a) virtual control δ (vehicle heading) which steers the subsystem (3.2) (vehicle position) to the origin, and (b) real control ω which render the dynamics of the subsystem (3.3) sufficiently faster than the subsystem (3.2) and stabilizes δ quickly to a desired virtual control, such that (3.2) becomes a slow subsystem and (3.3) becomes a fast subsystem in a singularly perturbed form. Note that this process is analogous to a human driver controlling the steering wheel (fast subsystem) to drive the vehicle (slow subsystem) to a desired pose in space.

3.3.1 Slow Subsystem and the Reference Heading

For the slow subsystem (3.2), consider a simple Lyapunov function candidate

$$V = \frac{1}{2}(r^2 + \phi^2) \tag{3.4}$$

Let the virtual control δ for the slow subsystem be

$$\delta = \arctan\left(-k_\phi \phi\right) \tag{3.5}$$

where k_ϕ is a positive constant. Then trajectory of (3.2) along (3.5) can be written as

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\left(\arctan\left(-k_\phi\phi\right)\right) \\ \frac{v}{r}\sin\left(\arctan\left(-k_\phi\phi\right)\right) \end{pmatrix} \tag{3.6}$$

Then the derivative of V along (3.6) becomes

$$\begin{aligned} \dot{V} &= r\dot{r} + \phi\dot{\phi} \\ &= -rv\cos\left(\arctan\left(-k_\phi\phi\right)\right) + \frac{v}{r}\,\phi\,\sin\left(\arctan\left(-k_\phi\phi\right)\right) \end{aligned}$$

which is strictly less than zero everywhere other than $r = 0$, since

$$\begin{aligned} \cos\left(\arctan\left(-k_\phi\phi\right)\right) &> 0, \qquad {}^\forall\phi \in (-\pi, \pi] \\ sgn(\arctan\left(-k_\phi\phi\right)) &= -sgn(\phi) \end{aligned}$$

and $r \geq 0$ and $v > 0$ by definition. Also, $\dot{\phi} < 0$ at boundary points $\{(r,\phi)|r > 0, \phi = \pi\}$. Thus the virtual control (3.5) steers the system (3.2) from arbitrary initial position in the set $[0,\infty) \times (-\pi, \pi]$ toward the origin. Note that since $\arctan(\cdot)$ is a smooth function and $\arctan 0 = 0$, we have $\delta \to 0$ as $\phi \to 0$ from (3.5), which implies the overall states $(r, \phi, \delta)^T$ also approaches the origin. Specifically, by choosing v to remove the singularity in r (e.g., by setting $v = f(r)$ such that $v = k_3 r$ in some neighborhood around $r = 0$, where k_3 is a positive constant), we can guarantee that the state always approaches origin. That is, for all $\epsilon > 0$, the closed ball $\bar{B}_\epsilon(0)$ is attractive in finite time.

The virtual control (3.5) can be understood as the reference heading of the vehicle obtained from current state ϕ. It characterizes a slow manifold (Fig. 3.2) that the fast dynamics of the vehicle heading will converge to.[2] The simple equation (3.5) enables us to design a set of reasonable and intuitive global manifolds leading to a given target, in the sense that the error coordinates r and ϕ always decrease smoothly.[3]

Note that it is the manifold, not a path, that is being planned here. In path-following, the controller will try to steer the vehicle back to a specified path when it deviates. But here, a new vehicle heading is recalculated with feedback, from manifold given by (3.5). This can be more robust and flexible than simple path following. The concept is closely related to the idea of *feedback motion planning*, which is a process of composing locally valid feedback policies that takes the system to the final destination (e.g. *Burridge et al.*

[2]When applied to a physical vehicle, care should be taken at the boundary of the manifold $\{(r,\phi)|r > 0, \phi = \pi\}$, where the reference heading in $\mathbb{R}^2$ can change suddenly under perturbation.

[3]The virtual control (3.5) is a smooth function on $(0,\infty) \times (-\pi, \pi)$.

(1999); *Zhang et al.* (2009); *Tedrake et al.* (2010)). This topic will be revisited in more detail in Chapter IV.

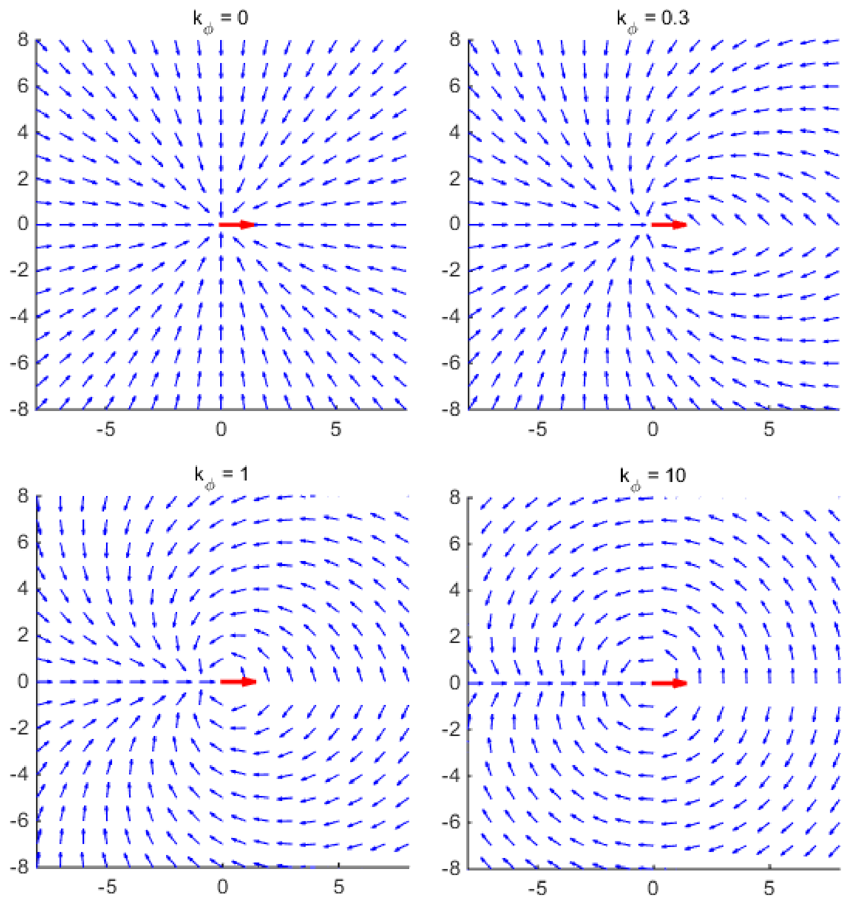


Figure 3.2: Design of slow manifold for the smooth control law

Design of slow manifold with k_ϕ, which is the ratio of the rate of change in ϕ to the rate of change in r. Target (red) is shown as a larger arrow at the center. Note $k_\phi = 0$ reduces the controller to pure waypoint-following, while $k_\phi \gg 0$ offers extreme scenario of pose-following where ϕ is reduced much faster than r.

Geometrically, the path given by the virtual control is the well-known Archimedean spiral (see *Weisstein*.) From (3.6), we have

$$\begin{aligned} \frac{\dot{\phi}}{\dot{r}} &= k_\phi \frac{\phi}{r} \\ \therefore \frac{\dot{\phi}}{\phi} &= k_\phi \frac{\dot{r}}{r} \end{aligned} \tag{3.7}$$

which implies that k_ϕ is the ratio of the rate of change in ϕ to the rate of change in r. The solution to (3.7) is

$$r = a\,\phi^{\frac{1}{k_\phi}} \tag{3.8}$$

with scaling factor $a = r_0/\phi_0{}^{\frac{1}{k_\phi}}$, where r_0 and ϕ_0 are initial conditions.

3.3.2 Fast Subsystem and Closed-Loop Steering

Now we develop a feedback control law for the steering. Let z denote the difference between the actual state δ and the desired property $\arctan(-k_\phi \phi)$, such that

$$z \equiv \delta - \arctan(-k_\phi \phi) \tag{3.9}$$

Calculation of $\dot{z}$ is straightforward. From (3.3) and (3.6),

$$\begin{aligned} \dot{z} &= \dot{\delta} - \frac{d}{dt}\arctan(-k_\phi \phi) \\ &= \dot{\phi} + \omega - \frac{-k_\phi}{1+(k_\phi \phi)^2}\dot{\phi} \\ &= (1 + \frac{k_\phi}{1+(k_\phi \phi)^2})\,\frac{v}{r}\sin(z + \arctan(-k_\phi \phi)) + \omega \end{aligned}$$

Now, let ω be

$$\omega = -\frac{v}{r}\,[\,k_\delta\, z + (1 + \frac{k_\phi}{1+(k_\phi \phi)^2})\sin(z + \arctan(-k_\phi \phi))] \tag{3.10}$$

with nonzero positive gain k_δ, so that

$$\dot{z} = -k_\delta \frac{v}{r} z \tag{3.11}$$

Furthermore, let $\tau \equiv \frac{r}{v}$, which is the minimum time needed for the vehicle to reach a goal, which is a relevant time scale for slow dynamics of (3.2). Then, with $k_\delta \gg 1$, we can treat $\epsilon \equiv \frac{\tau}{k_\delta}$ as a small time scale which ensures the fast dynamics to indeed be faster than the slow dynamics. That is, we have

$$\epsilon\,\dot{z} = -z \tag{3.12}$$

which is globally exponentially stable, and ensures that the fast subsystem quickly decays to the slow manifold. Numerical evaluation of the decay is shown in Fig. 3.3.

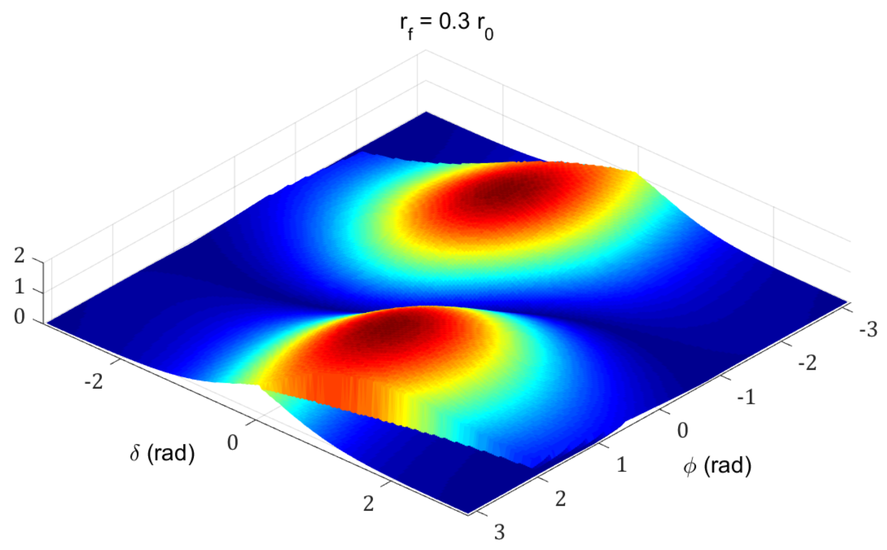


Figure 3.3: Numerical stability analysis for the smooth control law

Exponential decay of the heading error z under the control law (3.13) with $k_\phi = 1$ and $k_\delta = 3$. The vehicle starts with initial conditions of some nonzero $r_0 > 0$, $\phi \in (\pi, -\pi]$, and $\delta \in (-\pi, \pi]$, where $\max(z_0) \simeq 250\,(\mathrm{deg})$. Plot shows the value of z at $r_f = 0.3\,r_0$ against various values of initial conditions of ϕ and δ. The maximum value of $z_f < 1.9\,(\mathrm{deg})$.

In the original coordinates, the control law (3.10) can be written as

$$\omega = -\frac{v}{r}\left[\, k_\delta(\delta - \arctan(-k_\phi \phi)) + (1 + \frac{k_\phi}{1 + (k_\phi \phi)^2}) \sin\delta \,\right] \tag{3.13}$$

Thus, (3.13) is our control law for ω. Recall that we placed no restriction on v, other than to be nonzero positive.[4] That is, we essentially have v as a free variable.

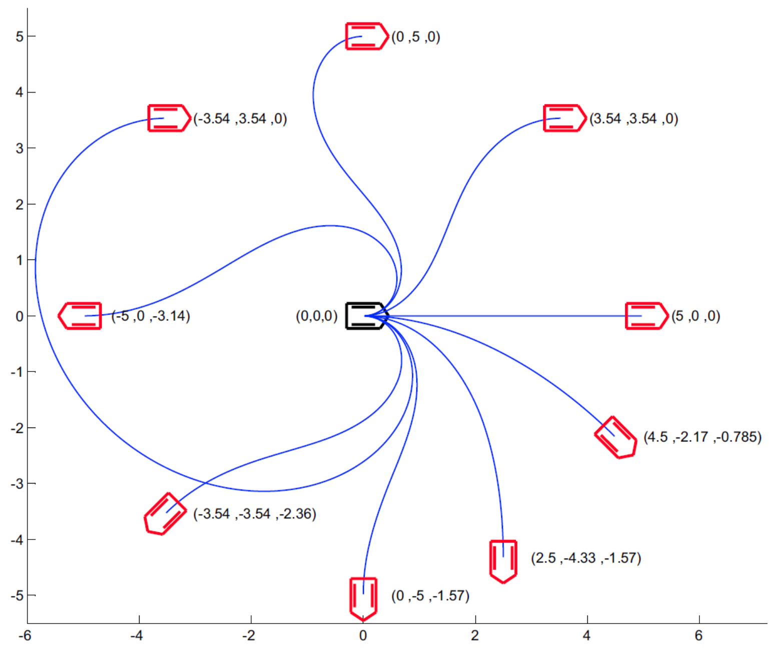
Also, ω is a linear function of v. Specifically, we have $\omega = \kappa(r, \phi, \delta)\, v$, where κ is the curvature of the path resulting from the proposed control law. Curvature of a path of a vehicle moving on a plane with linear velocity v and angular velocity ω is simply ω/v. We can write

$$\kappa = -\frac{1}{r}\left[\, k_\delta(\delta - \arctan(-k_\phi \phi)) + (1 + \frac{k_\phi}{1 + (k_\phi \phi)^2}) \sin\delta \,\right] \tag{3.14}$$

which implies that the shape of the path is not influenced by the choice of v.

Recall from the previous subsection that asymptotic convergence to the target depends on the choice of v. Namely, a small neighborhood of the origin in attractive in finite time if and only if $v \to 0$ as $r \to 0$. With such a choice, the control law is a solution to a so-called parking problem, stabilizing the vehicle at a specific target pose.

[4]To admit negative linear velocity, we can simply flip the orientation of the vehicle and the target simultaneously when the sign of v changes.



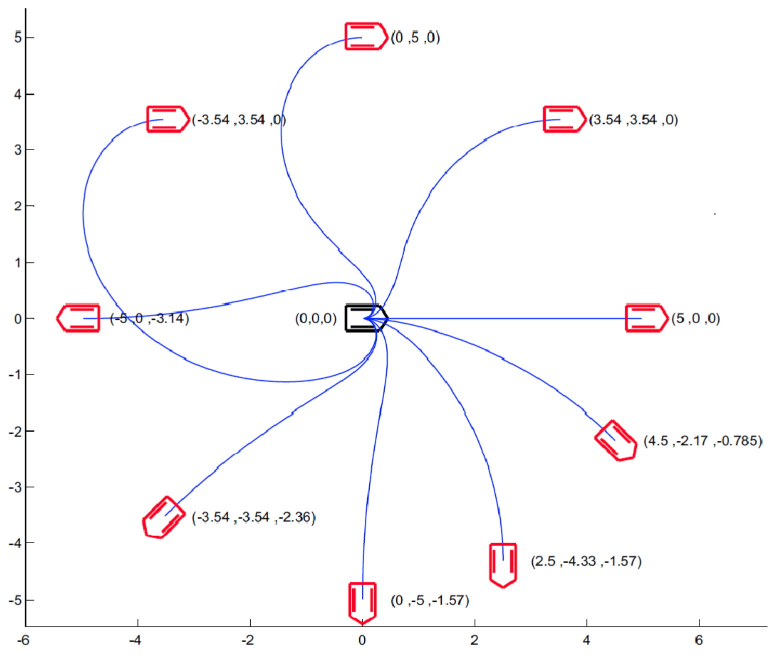


Figure 3.4: Example trajectories to targets under the smooth control law

Trajectories to various target poses under the control law. Initial pose is marked black at the center, and target poses are colored red. The coordinates of the poses are given as (x, y, θ), where θ is the orientation of the vehicle with respect to reference frame.
Top: Trajectories with constants $k_\phi = 1$ and $k_\delta = 3$.
Bottom: Trajectories with constants $k_\phi = 1$ and $k_\delta = 10$. Note that the vehicle converge to the slow manifold more quickly with higher value of k_δ.

The control law is also a solution to what can be called the passing problem: as the control law also guarantees that (r, ϕ, δ) approaches the origin with arbitrary nonzero positive v, the vehicle can arrive and pass through specific pose in space. It implies a straightforward implementation for a scenario following a path (given by a series of waypoint-poses).

Fig. 3.4, Top, shows some example trajectories of a unicycle under the proposed control law with constants $k_\phi = 1$ and $k_\delta = 3$. The vehicle can approach arbitrary motion targets with a smooth and intuitive curve. Fig. 3.4, Bottom, shows trajectories to the same motion targets, but with constants $k_\phi = 1$ and $k_\delta = 10$, which makes the vehicle converge much quicker to the reference heading.

3.3.3 Closed Loop Stability Analysis

For completeness, we close the section by sketching the overall proof. Let us introduce a coordinate transformation (3.9) to the system (3.1) augmented by the control law (3.13), such that

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\left(z + \arctan\left(-k_\phi \phi\right)\right) \\ \frac{v}{r}\sin\left(z + \arctan\left(-k_\phi \phi\right)\right) \end{pmatrix} \tag{3.15}$$

$$\epsilon\, \dot{z} = -z \tag{3.16}$$

with a small time parameter ϵ. The system is now in a standard singularly perturbed form, where (3.15) is the slow subsystem and (3.16) is the fast subsystem. The reduced system ($\epsilon = 0$)

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\left(\arctan\left(-k_\phi \phi\right)\right) \\ \frac{v}{r}\sin\left(\arctan\left(-k_\phi \phi\right)\right) \end{pmatrix}$$

globally approaches the origin, as can be seen in Fig. 3.2. The fast subsystem (3.16) is globally exponentially stable, such that it quickly decays to the reduced system, the slow manifold. Thus the system is guaranteed to converge to a small neighborhood of origin. [5]

3.4 Path Following via Static Motion Targets

In the previous section, we have established the convergence properties of the smooth control law (3.13). Now we build upon the result and show how graceful motion can be achieved, using a series of sparsely placed target poses as a path instruction. That

[5]The controller (3.5) is smooth on $(0, \infty) \times (-\pi, \pi)$ but is discontinuous on $\mathbf{R}^2$ at the boundary of the manifold. The controller (3.10) is smooth on $(0, \infty) \times (-\pi, \pi) \times (-\pi, \pi)$ but is discontinuous on $\mathbf{R}^3$ at the boundary of the manifold. Recall that the well-known Brockett's result (*Brockett*, 1983) states that it is not possible to find a smooth feedback control law for exact pose-stabilization for a unicycle. With our control law, we isolate the discontinuity to the boundary points of the manifold, and guarantee convergence to *any* small neighborhood of (but not exactly at) the origin.

is, a user instructs a sparse series of target poses, and the robot will generate full motor commands which guarantees graceful navigation via the targets[6]. When unacceptable target instructions (involving collisions, etc.) are given, they can easily be detected and rejected via model predictive simulation.

The solution described in this section is a useful form of navigation instruction with intermediate level of abstraction; the lowest level would be a series of motor commands, where the highest level would be specifying only the final goal in the map. In Chapter V, we extend the methods here with stochastic model predictive control to achieve graceful navigation for mobile robots in dynamic and uncertain environments.

3.4.1 Simple Linear Velocity Selector

In a path following scenario, perhaps the simplest approach for smooth navigation would be to adjust linear velocity of the vehicle as a function of path curvature. Note that the task becomes particularly easy with the proposed control law, since path curvature can be obtained directly from feedback (3.14) and linear velocity v is a free variable.

Specifically, we require $v \to 0$ as $\kappa \to \infty$, and $v \to v_{\text{max}}$ as $\kappa \to 0$, where v_{max} is imposed maximum linear velocity. A practical solution (not optimized) which satisfies the requirements is

$$v(\kappa) = v(r, \phi, \delta) = \frac{v_{\text{max}}}{1 + \beta |\kappa(r, \phi, \delta)|^{\lambda}} \tag{3.17}$$

where constants $\beta > 0$ and $\lambda \geq 1$ are design parameters.[7] Effect of design parameters are illustrated in Fig. 3.5

For the experiment in section 3.5, parameter values $\beta = 0.4$ and $\lambda = 2$ were used, with which $v(\kappa) \simeq 0.4\, v_{\text{max}}$ at $\kappa = 2$, i.e., when the radius of osculating circle is 0.5m. The angular velocity simply follows (3.13), that is, $\omega = \kappa v$ with v given by (3.17). With given parameters, analytic maximum for angular velocity becomes $\omega_{\text{max}} = \frac{\pi}{4}\, v_{\text{max}}$.

3.4.2 Heterogeneous Control and Human-like Driving Strategy

When presented with a single target, the control law (3.13) augmented with (3.17) allows the vehicle to navigate gracefully. But to navigate in more structured environment, multiple intermediate targets become necessary to reach a goal without collision. Then transitioning between targets becomes an inherent problem, since target switching intro-

[6]We avoid the term ‘waypoints’ since it implies the instructions are given as positions. Here the instructions are given as target poses.

[7]It may be practical to set some small v_{min} when far away from origin to promote faster turning motion when the curvature is very high.

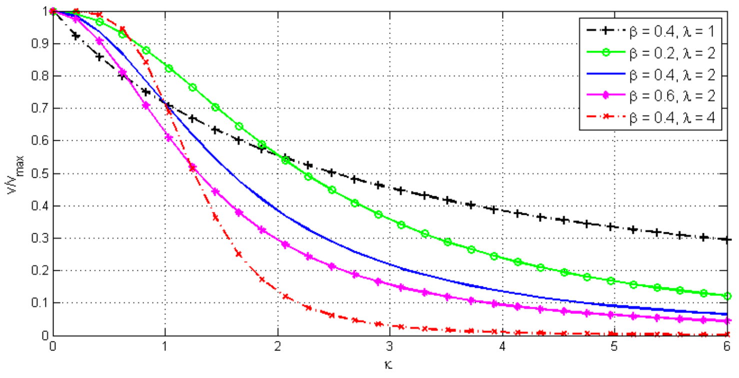


Figure 3.5: Simple linear velocity selector

Effect of design parameters $\beta > 0$ and $\lambda \geq 1$ on the curvature-based velocity rule (3.17) which has bell-shaped profile. Higher value of λ results in more sharply peaked curve, and higher value of β let the velocity drop more quickly as κ increases. These are design parameters that can be chosen freely to match the preference of the user.

duces discontinuity in vehicle state $(r, \phi, \delta)^T$, which may render derivatives of command signals arbitrarily large if the transition is instantaneous. Resulting high acceleration and jerk can lead to actuator overload (thus failure to execute a path) and discomfort to passengers.

Before we proceed further, let us briefly examine how human drivers navigate in an environment. When a human driver approaches a corner, the driver first examines how sharp the turn is and then reduces linear speed accordingly. At the corner, the driver would fix the linear velocity and change angular velocity smoothly to acquire desired heading, and then regain speed after passing the corner and the heading is stabilized. Considering how well a human driver can perform the task, it makes sense to adopt a similar procedure.

Given two consecutive targets $p_i = (r_i, \phi_i, \delta_i)^T$ and $p_{i+1} = (r_{i+1}, \phi_{i+1}, \delta_{i+1})^T$, we first slow down to a desired speed v_{p_i} before reaching p_i. The transition (analogous to turning a corner) is initiated at some threshold distance r_t away from p_i and lasts for a nonzero time interval τ_ω, so that the size of the derivatives become manageable. During the transition linear velocity is held constant and angular velocity transitions from initial angular velocity ω_r to $\omega = \kappa(p_{i+1}) \cdot v_{T_i}$. Finally, the vehicle will speed up again from v_{p_i} to $v(p_{i+1})$, following (3.17).

For this transition, we have developed a heterogeneous control scheme which is a variant of a method proposed in *Kuipers and Astrom* (1994). The idea is to gradually mix two heterogeneous control signals over transition interval τ, using a modified sigmoid function

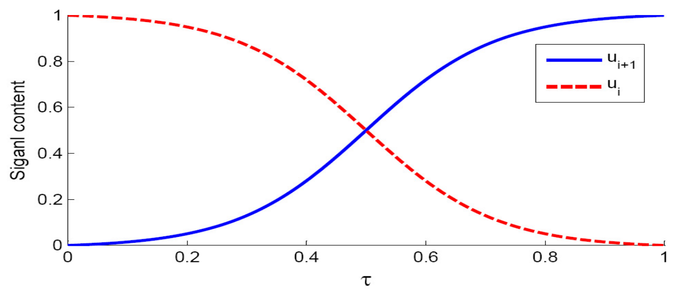


Figure 3.6: Heterogeneous control

Change in the signal content under (3.19), between two heterogeneous signals u_i and u_{i+1} during the transition period τ.

$\sigma_\tau(t)$ which is truncated and rescaled to domain $[0, \tau]$ and range $[0, 1]$:

$$\sigma_\tau(t) = \frac{1}{0.98}\left(\frac{1}{1 + e^{-9.2(t/\tau - 0.5)}} - 0.01\right) \tag{3.18}$$

Then the mixed signal during the transition from u_i to u_{i+1} is simply given by

$$u(t) = (1 - \sigma_\tau(t))\, u_i + \sigma_\tau(t) \cdot u_{i+1} \tag{3.19}$$

where $t \in [0, \tau]$ is a variable indicating the progress of transition. Fig. 3.6 shows the change in the signal content using the heterogeneous control.

It can be shown higher derivatives of (3.19) linearly depends on the magnitude of $\Delta u \equiv u_i - u_{i+1}$. In the proposed scenario, the magnitude of change in angular velocity command $\Delta\omega$ during the transition linearly depends on v_{p_i} since $\omega = \kappa\, v$. Thus v_{p_i} is an effective handle for the magnitude of the control signal ω and its higher derivatives. Desired linear velocity v_{p_i} given user-specified bounds can easily be obtained numerically via model predictive simulation.

The process can be formally described as forming a motion target $P_i \equiv (p_i^T, v_{p_i})^T$, which is a concatenated vector of target pose in space and the desired linear velocity. The desired linear velocity v_{p_i} is assigned to the target pose p_i based on the relationship between P_i and p_{i+1} under (3.13). Note that the process can be automated and does not require additional input from the user.

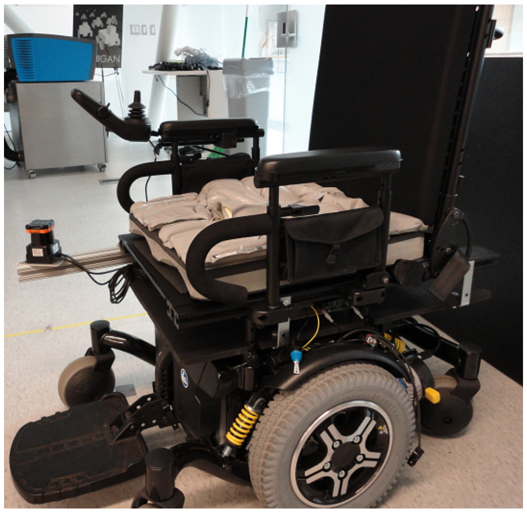


Figure 3.7: Differentially driven wheelchair robot.

(Best viewed in color) Snapshot of our differentially driven wheelchair robot, equipped with a laser range finder and an IMU, about 0.68m in length and 1.1m in width.

3.5 Evaluation on Physical Robot and Discussion

The proposed control law and the path following algorithm was tested on a physical robot (Fig. 3.7). The testbed is a differentially driven wheelchair robot equipped with a laser range finder and an IMU, 1.1m in length and 0.68m in width. Path instructions were given manually by specifying multiple target poses on a map generated via simultaneous localization and mapping (SLAM).

3.5.1 Results

A trajectory of the robot from a typical test run is shown in Fig. 3.8. The trajectory shown consists of a slow turn (lane change), a 90 degree turn and a final stop, typical actions required to navigate in office environment. Fig. 3.9 - 3.11 show the state of the robot along the trajectory.

In the run shown the robot traversed distance of 16.7m in 22.3 seconds. The robot was set to begin slowing down when it reaches the lookahead distance $r_l = 1.5$m from a motion target and commence target transition when it reaches the threshold distance $r_t = 1.0$m (about the size of the robot). The transition lasts over the interval $\tau_\omega = 1.3$s where the blending of angular velocity commands takes place. Numerical values of imposed bounds are $v_{\text{max}} = 1\,$m/s, $\omega_{\text{max}} = \frac{\pi}{4}\,$rad/s, $\dot{\omega}_{\text{max}} = 2.8\,$rad/s^2, and $\ddot{\omega}_{\text{max}} = 7.7\,$rad/s^3, all of which are satisfied over the entire trajectory.

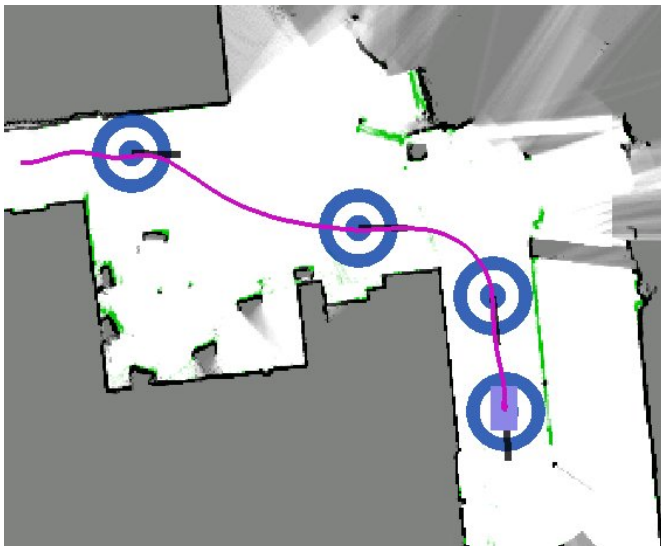


Figure 3.8: Example robot trajectory via static motion targets.

Trajectory of the robot traversing distance of 16.7m in 22.3 seconds. Motion targets are marked as concentric circles with protruding line segment indicating target orientation. Final position of the robot is shown as a purple rectangle.

For the implementation, care should be taken to choose appropriate values of threshold distance r_t, since $1/r$ term in the control law (3.13) can amplify the noise from the localization if v is constant. The issue can be resolved by either setting r_t sufficiently large (for intermediate motion targets) or by letting $v \sim r$, such that $v \to 0$ as $r \to 0$ which cancels each other out (for the final motion target).

Data was gathered at 20Hz via SLAM, which give us fairly accurate pose data over the trajectory. Velocity, acceleration and jerk were obtained from the pose data through standard numerical differentiation. To get reliable values for higher derivatives, however, some smoothing operations were necessary, and here local linear regression over a sliding time window of 0.5 second was applied. Since smoothing operations can significantly effect the calculation of the higher derivatives, we also show linear accelerations directly measured at 50Hz from an independent IMU, MicroStrain's *3DM-GX2 MicroStrain*, for qualitative validation. The shape and the level of the data corresponds well to the calculated value.

3.5.2 Discussion

The result demonstrates that the presented method can guide the robot quickly and comfortably through multiple motion targets, satisfying the requirement for graceful motion. The good performance of the kinematic control law and the path-following algorithm

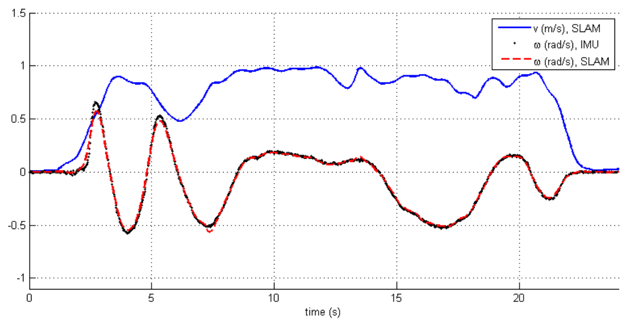


Figure 3.9: Velocities along the example robot trajectory

(Best viewed in color) Linear velocity v and angular velocity ω along the trajectory shown in Fig. 3.8. Angular velocity measurements from an IMU are also shown. The robot slows down as needed at $t = 4.7, 12.2, 17.1$s, where the target transition occurs. The velocities changes smoothly within imposed bounds.

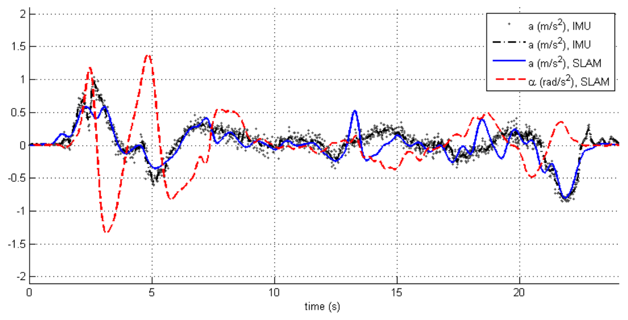


Figure 3.10: Accelerations along the example robot trajectory

(Best viewed in color) Linear acceleration $a = \dot{v}$, angular acceleration $\alpha = \dot{\omega}$, and linear acceleration measurements from the IMU along the trajectory shown in Fig. 3.8. Note that the differentiated and smoothed values of a closely tracks the independently measured IMU values.

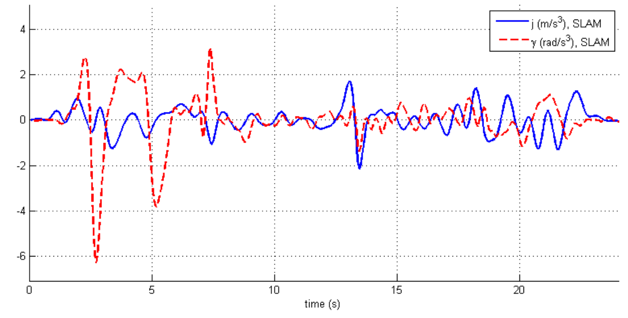


Figure 3.11: Jerks along the example robot trajectory

(Best viewed in color) Linear jerk $j = \ddot{v}$ and angular jerk $\gamma = \ddot{\omega}$ along the trajectory shown in Fig. 3.8. Both j and γ stays well within the imposed bounds, as can be expected from the smoothness of the trajectory.

does depends on its smooth nature. In general, kinematic controller works well only if underlying dynamic controller/actuator is sufficiently fast in achieving reference velocities calculated by the kinematic controller.[8] That is, the derivative of the velocity commands essentially amounts to required torque to satisfy the command. Thus smoothness is necessary for success of pure kinematic controllers.

In this Chapter, we have formulated the kinematic control law (3.13) and the pose-following algorithm for smooth and comfortable motion of unicycle-type robots. This work provides a concise and computationally very efficient solution to the local control problem of traveling between arbitrary poses quickly and comfortably in free space, which is a foundation for more general problem of navigating in dynamic and uncertain environment with static and dynamic obstacle avoidance. We also directly build on this and develop our stochastic model predictive control for graceful navigation in dynamic and uncertain environment, which is presented in Chapter V.

[8]See *Kim and Tsiotras* (2002) for experimental evaluation of some classical kinematic control laws on a physical robot.

CHAPTER IV

Non-holonomic Distance and Feedback Motion Planning via RRT*

[1] Here we present a non-holonomic distance function for unicycle-type vehicles, and use this distance function to extend the optimal path planner RRT* to handle nonholonomic constraints. The critical feature of our proposed distance function is that it is also a control-Lyapunov function. We show that this allows us to construct feedback policies that stabilize the system to a target pose, and to generate the optimal path that respects the non-holonomic constraints of the system via the non-holonomic RRT*. The composition of the Lyapunov function that is obtained as a result of this planning process provides stabilizing feedback and the cost-to-go to the final destination in the neighborhood of the planned path, adding much flexibility and robustness to the plan.

4.1 Proper Measure of Distance and Optimal Path Planning

For efficient and intelligent navigation, a mobile robot planning a path must have a good measure of how far a given pose (position and orientation) in navigable space is to another pose. For mobile robots with non-holonomic constraints (e.g., robots that cannot move sideways), the widely-used Euclidean distance in Cartesian coordinates is clearly not a sufficient metric for this, since it fails to capture the constraints and does not reflect the true cost-to-go of the system. (Fig. 4.1.)

This incompatibility of the Euclidean distance for non-holonomic systems is a well-known problem. In fact, it has been shown that for sampling-based planners such as RRT (*LaValle*, 1998) the space is efficiently explored only when this distance function reflects the true cost-to-go (*Cheng and LaValle*, 2001). But this has been getting more attention recently (e.g., see *Karaman and Frazzoli* (2013); *Webb and Berg* (2012); *Goretkin et al.*

[1]This chapter is a revised (and expanded) presentation of *Park and Kuipers* (2015).

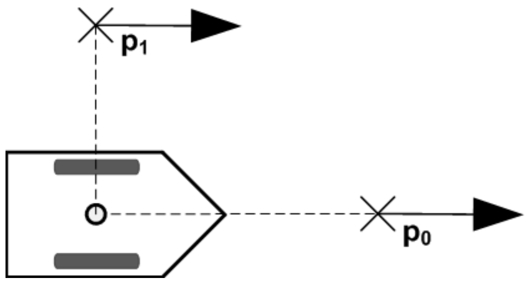


Figure 4.1: Incompatibility of Euclidean distance for non-holonomic systems.

Consider two poses p_0 and p_1. Although p_1 is nearer the robot in Euclidean distance, it is harder to get to due to differential constraints. In this paper, we propose a directed distance function applicable to unicycle-type vehicles, that properly reflects the true cost-to-go of the system under the non-holonomic constraint.

(2013); *Palmieri and Arras* (2014)) since the introduction of RRT* (*Karaman and Frazzoli*, 2011), the sampling-based motion planner that guarantees asymptotic optimality, which critically requires a proper distance function. In the original formulation (*Karaman and Frazzoli*, 2011) Euclidean metric was used to measure distance, which is a proper metric only for holohomic systems.

Several attempts have been made to extend RRT* to non-holonomic systems, where many recent works focus on some specific form of steering function that connects two states and the cost-to-go under the control. In *Webb and Berg* (2012), the distance between a pair of states is measured based on a fixed-final-state/free-final-time controller and a cost function on time and control effort. In *Goretkin et al.* (2013) and *Perez et al.* (2012), LQR-based cost functions are used to measure distance. Both *Webb and Berg* (2012) and LQR-based methods *Goretkin et al.* (2013) *Perez et al.* (2012) rely on local linearization of dynamics for non-linear systems. In *Palmieri and Arras* (2014), an approximation of the path-integrated cost is used as the distance metric, where the approximation is learned from simulated trajectories of a differential-drive vehicle under a stabilizing control law. A different approach was recently proposed in *Karaman and Frazzoli* (2013) by the authors of the original RRT*, which does not concern the steering or the cost-to-go. The key idea of *Karaman and Frazzoli* (2013) is to use a weighted Euclidean box which is narrower in the direction the motion is constrained, rather than an Euclidean ball, when identifying nearby neighbors, thus resulting in less number of bad candidate connections and improved efficiency of the RRT*.

In this paper, we present a parameterized closed-form distance function from a pose (position and orientation) to a target pose for unicycle-type vehicles (e.g. differentially driven mobile robots), and an optimal sampling-based planner using the distance function (the non-holonomic RRT*), where the free parameters in the distance function controls the shape of the resulting path. The critical feature of our proposed distance function is that it is also a control-Lyapunov function (CLF) for the system, which makes it a natural measure of cost-to-go. We show that it is straightforward to develop feedback policies given the distance function, and develop a non-holonomic RRT* based on this distance and the derived policy. With our approach, the non-holonomic RRT* not only generates an optimal path but also a composition of Lyapunov functions (so-called funnels (*LaValle*, 2006; *Mason and Salisbury Jr*, 1985; *Burridge et al.*, 1999; *Tedrake et al.*, 2010)) that provides stabilizing feedback and the cost-to-go to the final destination in the neighborhood of the planned path, adding much flexibility and robustness to the plan.

4.2 Non-holonomic Distance Functions, Vector Fields, and Funnels

We define our distance function on a smooth manifold defined by the egocentric polar coordinates (*Park and Kuipers*, 2011). This egocentric polar coordinate system (Fig. 3.1) describes the relative location of the target pose observed from a vehicle with radial distance r, orientation of the target ϕ, and orientation of the vehicle δ, where angles are measured from the line of sight vector from the vehicle to the target. Formally, we write

$$\mathscr{T}_{\mathrm{p}_0} : (x, y, \theta)^T \mapsto (r, \phi, \delta)^T \tag{4.1}$$

to denote the mapping from the usual Cartesian coordinates to the egocentric polar coordinates, where p_0 is the pose of the target and $\mathrm{p} = (x, y, \theta)^T$ is the pose of the vehicle.

Recall from Chapter III that the kinematics of the vehicle in the egocentric polar coordinates can be written as

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -v\cos\delta \\ \frac{v}{r}\sin\delta \end{pmatrix} \tag{4.2}$$

$$\dot{\delta} = \frac{v}{r}\sin\delta + \omega \tag{4.3}$$

where (4.2) describes the dynamics of the slow subsystem in position subspace, and (4.3) describes the dynamics of the fast subsystem in heading (steering) subspace. Note that in the slow subsystem (4.2), the heading δ is a (virtual) control.

In the position subspace, we use the following weighted norm to measure how far a point $(r, \phi)^T$ is to the origin (target pose):

$$l^{-}(r, \phi) = \sqrt{r^2 + k_\phi^2\,\phi^2} \tag{4.4}$$

where k_ϕ is a positive constant that represent the weight of ϕ with respect to r. Note that this is a metric in polar coordinates, and the orientation of the target pose is incorporated in ϕ.

It is trivial to show that (4.4) is a CLF for the virtual control δ, as there always exists some heading that reduces this positive definite function. For example, simply setting the desired heading at $\delta^* = 0$ (which points directly to target pose) results in the classic turn-travel-turn approach. Here we show the origin is Lyapunov-stable under the following feedback examples, assuming non-zero positive velocity v:

$$\begin{aligned} \delta_s^*(\phi) &= \operatorname{atan}(-k_\phi \phi) && (4.5) \\ \delta_g^*(r,\phi) &= \operatorname{atan}(-k_\phi^2 \phi / r^2) && (4.6) \end{aligned}$$

where (4.5) is the heading that reduces r and ϕ very smoothly at ratio $\dot{\phi}/\phi = k_\phi \dot{r}/r$ (see Chapter III), and (4.6) is the gradient of (4.4) along (4.2). As will be shown in the results, these are very distinct and useful steering strategies, with (4.5) generating very smooth curves and (4.6) generating curves that quickly approaches the target pose and then aligns to the target orientation. Each of these feedback control laws for the heading (4.5)-(4.6), which specifies a heading vector at every point in the position space by construction, describes a stabilizing *vector field.* This vector field encodes a collection of paths converging to the target over the entire space, with tangents of the paths aligned with the vector field (Fig. 4.2). We use this to connect two nodes in the RRT tree, in Extend() and Steer() processes as described in Section 4.3.

To show the origin is Lyapunov stable, we need to check the sign of the derivative of the positive definite function (4.4) along the position dynamics (4.2) under the control (4.5) - (4.6). We have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(l^-(\cdot)^2) &= 2r\dot{r} + 2\phi\dot{\phi} \\ &= -2rv\cos\left(\delta_{[\cdot]}^*(\cdot)\right) + \frac{2v}{r}\,\phi\,\sin\left(\delta_{[\cdot]}^*(\cdot)\right) \end{aligned}$$

strictly less than zero everywhere other than $r = 0$, since

$$\begin{aligned} \cos\left(\delta_{[\cdot]}^*(\cdot)\right) &> 0, & \because\ \delta_{[\cdot]}^*(\cdot) &\in (-\pi/2, \pi/2) \\ sgn(\delta_{[\cdot]}^*(\cdot)) &= -sgn(\phi), & \forall \phi &\in (-\infty, \infty) \end{aligned} \qquad (4.7)$$

due to property of $\operatorname{atan}(\cdot)$ and $r \geq 0$ and $v > 0$ by definition. Thus $\frac{\mathrm{d}}{\mathrm{d}t} l^-(\cdot)^2$ is also negative definite, and the origin (target pose) is Lyapunov stable under $\delta_{[\cdot]}^*(\cdot)$.

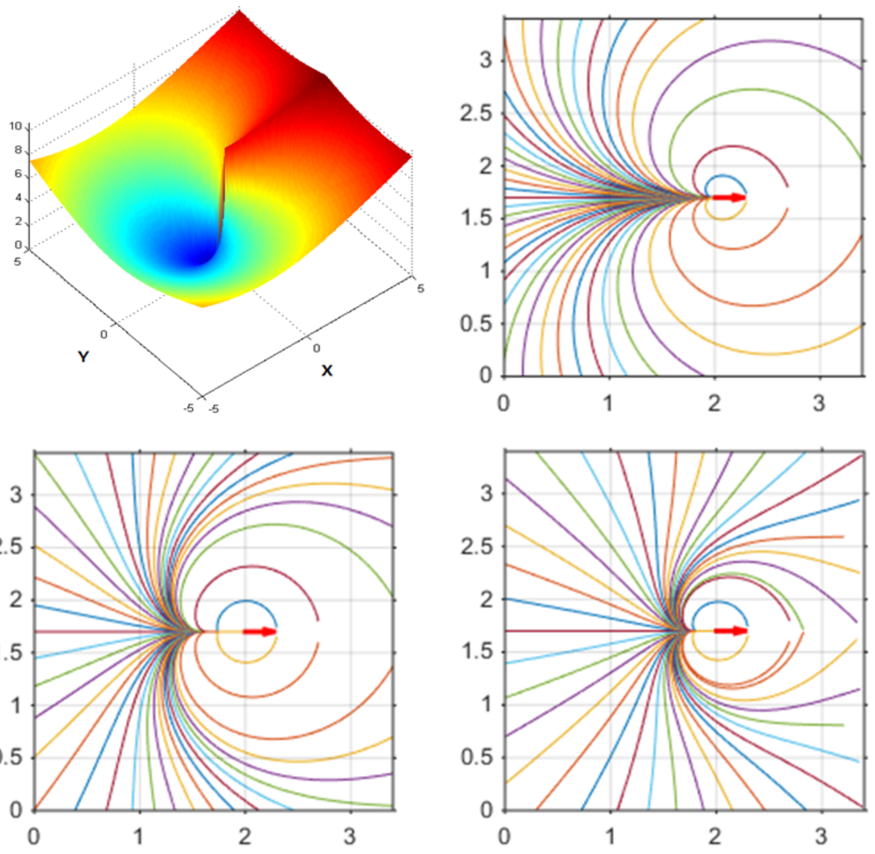


Figure 4.2: Visualization of a non-holonomic distance function, and example paths tracing user-specified vector fields.

This figure visualize a (reduced) non-holonomic distance function, and example paths tracing the user-specified vector fields. These stabilizing vector fields and the shape of the resulting paths can be tuned to user's preference with a positive weight k_ϕ, where higher k_ϕ results in quicker reduction in angular coordinate ϕ. Note that with (4.5) (bottom row) the paths always curves in smoothly to the target pose regardless of initial position, but with (4.6) (top right) large portion of the paths shown are almost straight lines until it is near the target and then elbows in to the target orientation.
Top Left: Illustration of distance from various positions to a target pose at $(0, 0, 0)^T$ (facing to the positive x-axis), assuming the heading is aligned with the specified vector field, so the distance function (4.9) reduces to (4.4). Note the singularity at origin and the high cost at positions along positive x-axis, where the the vehicle have to make a large arc to move to the target pose ($k_\phi = 1.0$).
Top Right: Smooth curves following (4.5), with $\dot{\phi}/\phi = k_\phi \dot{r}/r$ ($k_\phi = 1.5$). Target pose is depicted with a red arrow at $(2, 1.7, 0)^T$.
Bottom Left: Sample paths following (4.6), the gradient of (4.9) along the dynamics (4.2) ($k_\phi = 0.8$).
Bottom Right: Sample paths following (4.6) ($k_\phi = 1.5$). Note that the angular coordinate ϕ is reduced much quicker compared to figure on top right.

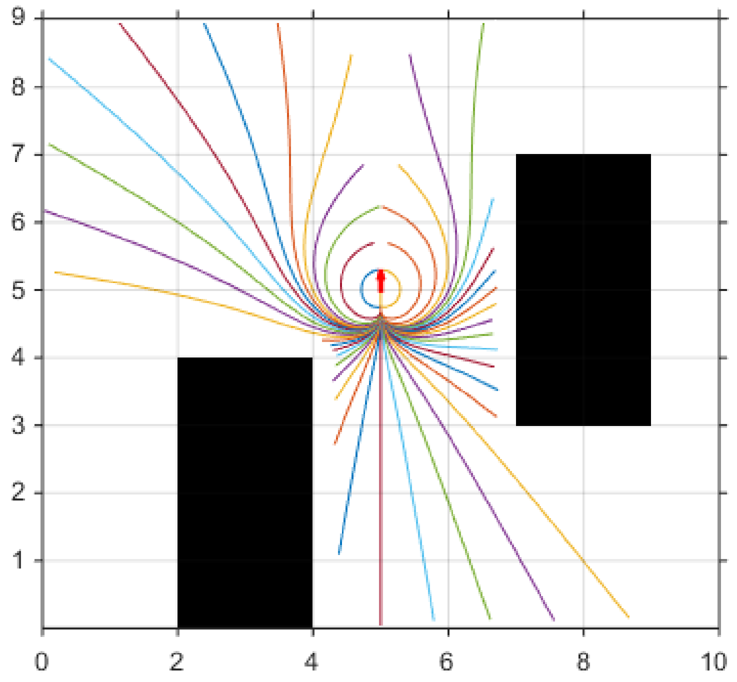


Figure 4.3: Estimating the domain of attraction in structured environments.

The domain of attraction (the mouth of the funnel) around a target pose for a unicycle can be estimated by tracing the vector field in environment with obstacles. Here a target pose (red arrow facing upward) is at $(5, 5, \pi/2)^T$, and the paths follow (4.6) with $k_\phi = 1.2$.

These vector fields are examples of so-called *funnels* in feedback motion planning (*LaValle*, 2006). A funnel (*Mason and Salisbury Jr*, 1985; *Burridge et al.*, 1999) can be described as a (locally) valid feedback policy or a Lyapunov function, which takes a broad set of initial conditions to a local subgoal. *Feedback motion planning* is a process of finding a sequential composition of funnels that takes the system to the final destination, which can be more robust and flexible than simple path planning. See *Lindemann and LaValle* (2009) *Zhang et al.* (2009) for the use of vector fields as funnels; *Tedrake et al.* (2010) is an excellent example of feedback motion planning based on local linearizations and LQR policies. Existing approaches tend to focus on finding local controllers due to usual difficulty in finding a general Lyapunov function. One of the contribution of this thesis is the presentation of (4.9), which is a control-Lyapunov function for unicycle-type vehicles, which is our funnel.

The domain of attraction for this Lyapunov function (i.e. the mouth of the funnel) is the entire configuration space for a unicycle in completely free space, but in cluttered environments (or for curvature-constrained vehicles) it needs to be estimated numerically. We can make a conservative estimate by tracing the vector field, as shown in Fig. 4.3.

Given the stabilizing vector field in position space characterized by the desired heading $\delta^*_{[\cdot]}(\cdot)$, we can define the heading error with respect to the vector field as

$$\delta_e(r, \phi, \delta) = |\delta - \delta^*_{[\cdot]}(\cdot)| \tag{4.8}$$

which is the distance between the robot orientation and the desired heading $\delta^*_{[\cdot]}(\cdot)$. With this we can define a CLF in the full configuration space, where linear and angular velocities are treated as control inputs,

$$\begin{aligned} l(r, \phi, \delta) &= l^{-}(r, \phi) + k_\delta\, \delta_e(r, \phi, \delta) \\ &= \sqrt{r^2 + k_\phi^2\, \phi^2} + k_\delta |\delta - \delta^*_{[\cdot]}(\cdot)| \end{aligned} \tag{4.9}$$

where k_ϕ and k_δ are positive constants. Note that the choice of k_ϕ controls the shape of the local path segments via $\delta^*_{[\cdot]}(\cdot)$, and k_ϕ and k_δ influences the distance definition, i.e. how far the origin is from a pose.

This is a CLF for kinematic inputs since there always exists some linear and angular velocity (v, ω) that reduces this positive definite function.[2] [3] Note that this is a weighted L1 norm of the distances in position (4.4) and heading (4.8). The choice of L1 norm is motivated by the fact that minimizing L1 norm tends to reduce one of the terms first, while minimizing L2 norm tends to reduce all terms simultaneously. As will be seen in the Section 4.3, this choice of L1 norm tends to create very smooth path, as it strongly motivates the planner to compose funnels so that they have minimal heading error at joints.

Formally, we define a directed distance from a pose $\mathrm{p} = (x, y, \theta)^T$ to a target pose p_0, applicable to unicycle-type vehicles with non-holonomic constraints via (4.1) and (4.9):

$$\text{Dist}(p, p_0) = l(\mathscr{T}_{\mathrm{p}_0}((x, y, \theta)^T)) \tag{4.10}$$

Recall that this measures the distance assuming vehicle is moving forward. Flipping both p and p_0 by 180 degrees yields the distance for backward motion. This function is positive definite but is not symmetric, i.e. Dist(p, p_0) $\neq$ Dist(p_0, p). Also, this distance function is similar to the radial distance r when $r \gg 1$, while promoting alignment to target orientation as $r \to 0$.

[2]A unicycle can always rotate in place to align to the vector field, and move forward if the heading is well aligned.

[3]This is the CLF used to derive the smooth control law in Chapter III. The vector fields represent the virtual control for the position subsystem. The key idea is to guarantee that the heading error (4.8) is reduced sufficiently faster than the position error (4.4) (via two-time scale decomposition *Khalil* (2002)) so that the vehicle always converges to the virtual control, and ultimately to a small neighborhood of target pose. See Chapter III for the detailed derivation.

In the next section, we use the heading control $\delta^*_{[\cdot]}(\cdot)$ as our steering function, and (4.9) as our distance and cost function for non-holonomic RRT* for unicycle-type vehicles.

4.3 Non-holonomic RRT* for Unicycles

RRT* (*Karaman and Frazzoli*, 2011) is an incremental, sampling based planner with guaranteed asymptotic optimality, originally developed for holonomic systems. The algorithm (Alg. 1-3) is an extension of the RRT* for unicycle-type vehicles, using the steering and the distance function developed in the previous section. It solves the optimal path planning problem by growing a tree $\mathcal{T} = (V, E)$ with vertex set V of poses connected by edges E of feasible path segments to find a path that connects exactly to the destination pose with minimum cost, using the set of basic procedures described below. X is the configuration space of the vehicle, and $x : [0, 1] \to X$ is a path in the configuration space. The notations generally follows the RRT* algorithm presented in *Karaman et al.* (2011).

Sampling: The Sample() process samples a pose $z_{\text{rand}} \in X_{\text{free}}$ from obstacle-free region of the configuration space. The sampling is random with a goal bias, with which the destination pose is sampled to ensure the path exactly connects to the destination pose.

Distance: Dist(z, z_0) as defined in (4.10) returns the directed distance from a *pose* z to a *target pose* z_0.

Cost: The cost function $c(z, z_0)$ measures the cost of the path from a pose z to a pose z_0. In this paper, we are finding the minimum-distance path, so $c(z, z_0) = \text{Dist}(z, z_0)$. Cost($v$) returns the accumulated cost of a node $v \in V$ in the tree from the root.

Nearest Neighbor: Given a pose $z \in X$ and the tree $\mathcal{T} = (V, E)$, $v = \text{Nearest}(\mathcal{T}, z)$ returns the node in the tree where the distance from the node to the pose Dist(v, z) is minimum.

Near-by nodes: Given a pose z, tree $\mathcal{T} = (V, E)$, and a number n, we have

$$\text{NearTo}(\mathcal{T}, z, n) \equiv \{v \in V \mid \text{Dist}(v, z) \leq L(n)\} \tag{4.11}$$

where $L(n) = \gamma(\frac{log(n)}{n})^{(1/d)}$ with constant γ and dimension of the space d (*Karaman and Frazzoli*, 2011), where we have $d = 3$ for our configuration space. This returns set of nodes from which the distance *to* the pose z is small. Similarly,

$$\text{NearFrom}(\mathcal{T}, z, n) \equiv \{v \in V \mid \text{Dist}(z, v) \leq L(n)\} \tag{4.12}$$

returns the set of nodes that is near *from* the pose z. This distinction between NearTo() and NearFrom() is necessary due to the use of directed distance.

Algorithm 1: $\mathcal{T} = (V, E) \leftarrow$ Nonholonomic RRT$^*(z_{\text{init}})$

```
1  $\mathcal{T} \leftarrow$ InitializeTree();
2  $\mathcal{T} \leftarrow$ InsertNode($\emptyset, z_{\text{init}}, \mathcal{T}$);
3  for $i = 1$ to $N$ do
4      $z_{\text{rand}} \leftarrow$ Sample($i$);
5      $z_{\text{nearest}} \leftarrow$ Nearest($\mathcal{T}, z_{\text{rand}}$);
6      $(z_{\text{new}}, x_{\text{new}}) \leftarrow$ Extend($z_{\text{nearest}}, z_{\text{rand}}, \epsilon$);
7      if ObstacleFree($x_{new}$) then
8          $Z_{\text{nearTo}} \leftarrow$ NearTo($\mathcal{T}, z_{\text{new}}, |V|$);
9          $z_{\text{min}} \leftarrow$ ChooseParent($Z_{\text{nearTo}}, z_{\text{nearest}}, z_{\text{new}}$);
10         $\mathcal{T} \leftarrow$ InsertNode($z_{\text{min}}, z_{\text{new}}, \mathcal{T}$);
11         $Z_{\text{nearFrom}} \leftarrow$ NearFrom($\mathcal{T}, z_{\text{new}}, |V|$);
12         $\mathcal{T} \leftarrow$ ReWire($\mathcal{T}, Z_{\text{nearFrom}}, z_{\text{min}}, z_{\text{new}}$);
13 return $\mathcal{T}$
```

Algorithm 2: $z_{\text{min}} \leftarrow$ ChooseParent($Z_{\text{nearTo}}, z_{\text{nearest}}, z_{\text{new}}$)

```
1  $z_{\text{min}} \leftarrow z_{\text{nearest}}$;
2  $c_{\text{min}} \leftarrow$ Cost($z_{\text{nearest}}$) $+ c(z_{\text{nearest}}, z_{\text{new}})$;
3  for $z_{near} \in Z_{nearTo}$ do
4      $x' \leftarrow$ Steer($z_{\text{near}}, z_{\text{new}}$);
5      if ObstacleFree($x'$) then
6          $c' =$ Cost($z_{\text{near}}$) $+ c(z_{\text{near}}, z_{\text{new}})$;
7          if $c' <$ Cost($z_{new}$) $and$ $c' < c_{min}$ then
8              $z_{\text{min}} \leftarrow z_{\text{near}}$;
9              $c_{\text{min}} \leftarrow c'$;
10 return $z_{min}$
```

Algorithm 3: $\mathcal{T} \leftarrow$ ReWire($\mathcal{T}, Z_{\text{nearFrom}}, z_{\text{min}}, z_{\text{new}}$)

```
1  for $z_{near} \in Z_{nearFrom} \setminus \{z_{min}\}$ do
2      $x' \leftarrow$ Steer($z_{\text{new}}, z_{\text{near}}$);
3      if ObstacleFree($x'$) and
       Cost($z_{new}$) $+ c(z_{new}, z_{near}) <$ Cost($z_{near}$) then
4          $\mathcal{T} \leftarrow$ ReConnect($z_{\text{new}}, z_{\text{near}}, \mathcal{T}$);
5  return $\mathcal{T}$
```

Steering: We assume the vehicle follows the vector field, e.g. the feedback control on the heading $\delta^*_{[\cdot]}(\cdot)$ described in (4.5) and (4.6).[4] We have $x = \text{Steer}(z, z_0)$ which generate a path segment x that starts from z and end exactly at z_0, and $(z_{\text{new}}, x_{\text{new}}) = \text{Extend}(z, z_0, \epsilon)$ starts from z and extend the path toward z_0 until z_0 is reached or the distance covered is ϵ, and returns the new sample z_{new} at the end of the extension. Extend() is a standard RRT procedure for exploration. In our experiments, we obtained best results when we used Extend() for exploration (not forcing exact connection to a target sample) and Steer() for choosing the best parent node and to rewire graph, as described in Alg. 1-3.

Collision Checking: The ObstacelFree(x) function checks whether a path x lies within the obstacle-free region of the configuration space, considering the size and the shape of the robot. Note that any mission-critical constraints (e.g. path curvature, minimum clearance, etc.) that needs to be checked can be added here.

Node Insertion: Given the current tree $\mathcal{T} = (V, E)$ and a node $v \in V$, the function InsertNode(v, z, $\mathcal{T}$) adds z to V and create an edge to v from z.

With theses basic processes, the non-holonomic RRT* explores the configuration space by sampling and extending as the classic RRT (*LaValle*, 1998) (Alg.1), and considers all nearby nodes of a sample to choose the best parent node and rewires the graph to streamline the tree as the RRT* (Alg.2-3) where ChooseParent() and ReWire() processes guarantee asymptotic optimality. We provide proper measure of distance and exact steering for unicycle-type vehicles, which are necessary components for these processes.

4.4 Feedback Motion Planning via RRT*

In this section we demonstrate the non-holonomic RRT* with varying steering functions (Fig. 4.4-4.8). Recall that the distance function also depends on the steering function, so that the planning is integrated with the control. Overall, the algorithm is able to accommodate the two example controls (4.5)-(4.6) very well, with (4.5) leading to very smooth and elegant paths, while the use of (4.6) generates more aggressive composition of L-shaped curves.

4.4.1 Results

Fig. 4.4 illustrates the incremental sampling process, using (4.5) with weights $k_\phi = 1.2$ and $k_\delta = 3.0$. Notice that the quality of the path, especially the smoothness of the curve,

[4]Assuming initial condition $\delta_e = 0$, the heading control $\delta^*_{[\cdot]}(\cdot)$ can be exactly followed with $\omega = \dot{\delta}^* - \frac{v}{r}\sin\delta^*$ via (4.3), where the positive linear velocity v can be chosen so that the accelerations and velocities are bounded. See footnote 2 in the previous section for discussion on more general controllers.

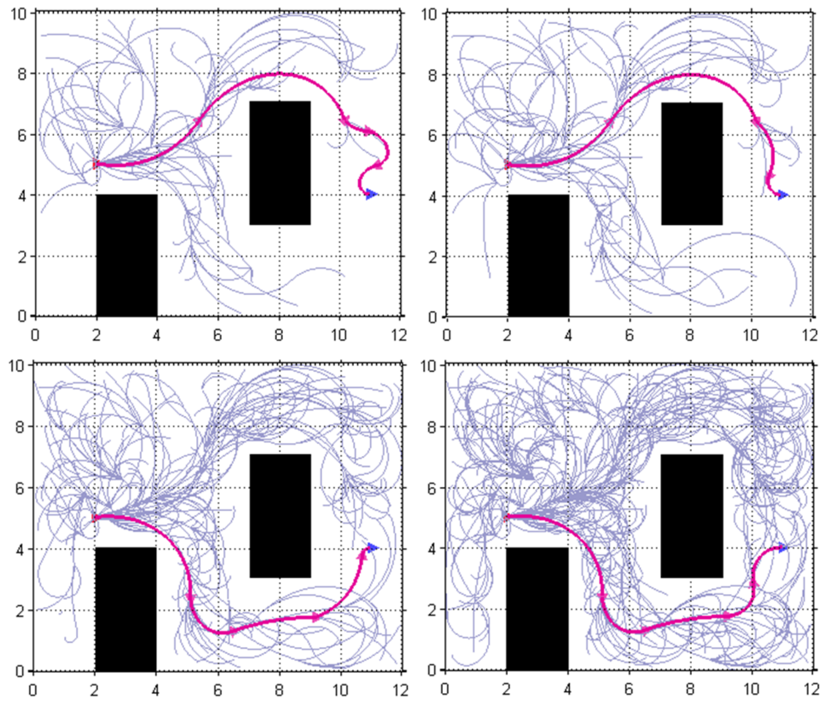


Figure 4.4: Incremental sampling process by the non-holonomic RRT*

Illustration of an incremental sampling process performed by the proposed algorithm, using the distance function (4.10) via (4.5), with weights $k_\phi = 1.2$, $k_\delta = 3.0$. **From top left, reading order**: The tree constructed by the non-holonomic RRT^* with 112, 163, 355, and 731 vertices. The algorithm first finds a path across the larger open space in top area, but eventually finds shorter distance path through bottom, meanwhile improving the overall smoothness of the path.

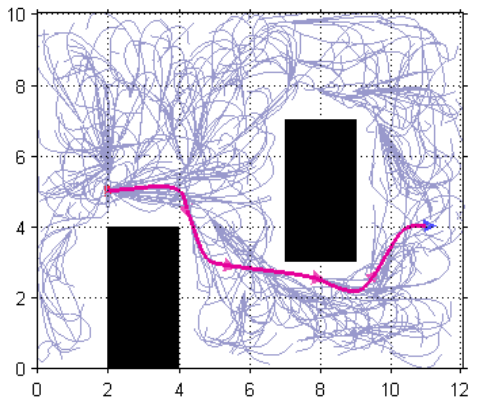


Figure 4.5: An example path found by the non-holonomic RRT*

The minimum-distance path in the tree of 907 vertices, with the proposed algorithm using the distance function (4.10) via (4.6), with weights $k_\phi = 1.2$ and $k_\delta = 3.0$. Note that with (4.6), the path is composed of L-shaped segments, which quickly moves toward a target pose (the next node) to reduce the radial distance and then elbows in to the target orientation.

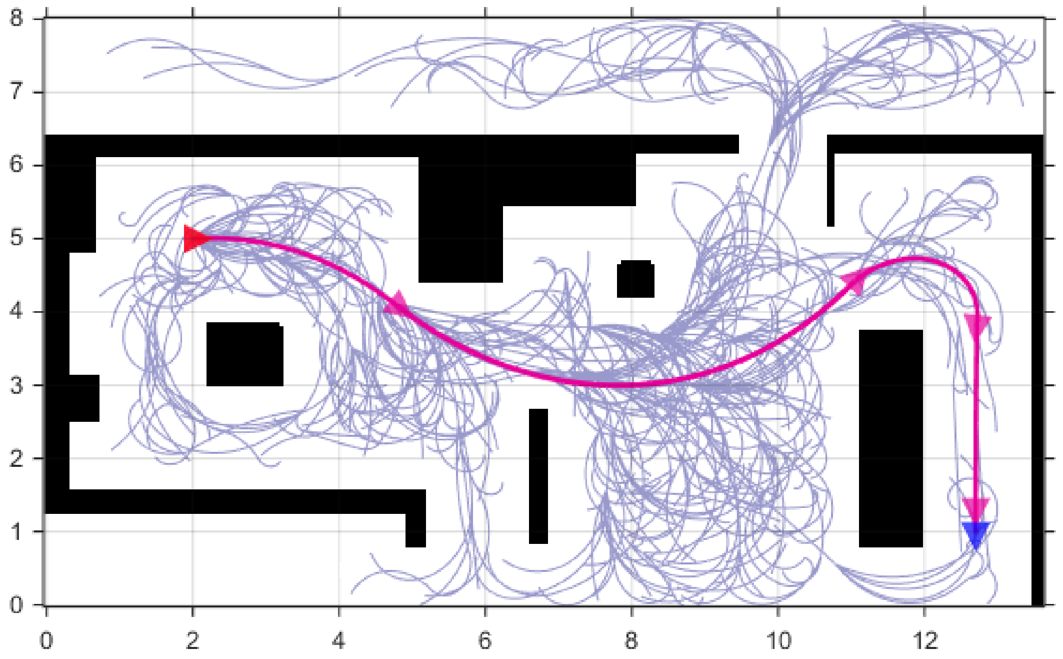


Figure 4.6: Example minimum-distance path found by the non-holonomic RRT*

(Best viewed in color) An example minimum-distance path (bold line) found by our non-holonomic RRT* after 1000 vertices, using the proposed distance function (4.10) and the exact steering (4.5), with constants $k_\phi = 1.2$ and $k_\delta = 3.0$. The path exactly connects the starting pose at top left facing right (red triangle) and destination pose at bottom right facing downward (blue triangle) in a cluttered 14m$\times$8m office environment. The vertices explored (grey) are also shown. Our algorithm is able to generate highly smooth and intuitive paths without requiring any extra processing. Note that it rejects candidate paths with shorter Euclidean distance but is harder to get to considering the constraints of the system, such as paths that cross below the rightmost obstacle and reach the destination facing in the wrong direction.

gradually improves as the number of vertices increase. The high value of k_δ heavily penalizes heading error between the sampled pose and the subsequent path segment, which enforces the overall smoothness. k_ϕ determines the shape of local path segments via steering. For comparison, Fig. 4.5 shows an example path obtained using (4.6) with the same parameters $k_\phi = 1.2$, $k_\delta = 3.0$. Fig. 4.4 and Fig. 4.5 shows both (4.5) and (4.6) produce satisfactory, yet qualitatively distinct, paths.

Fig. 4.6 - 4.8 shows an example result that demonstrates the performance of the proposed method. Fig. 4.6 shows the minimum-distance path found (after 1000 vertices) in cluttered office environment, using (4.5) for the steering and for the distance definition. Fig. 4.7 shows that shorter, smoother paths are selected over longer, less smooth ones, as desired.

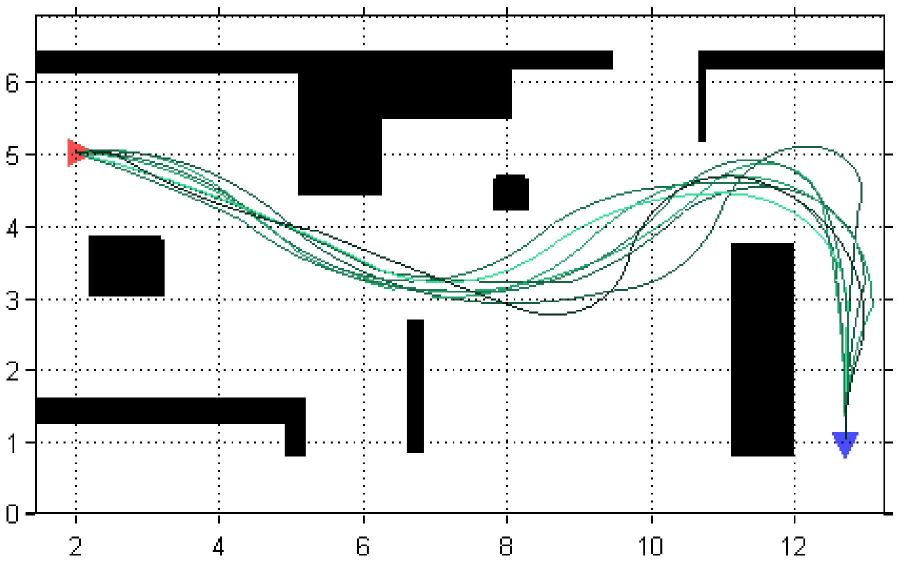


Figure 4.7: Multiple (suboptimal) solutions obtained over several runs of RRT*.

(Best viewed in color) Green to black color gradient indicates low to high cost of the path. Shorter, smoother paths have lower cost than longer, less smooth ones as desired.

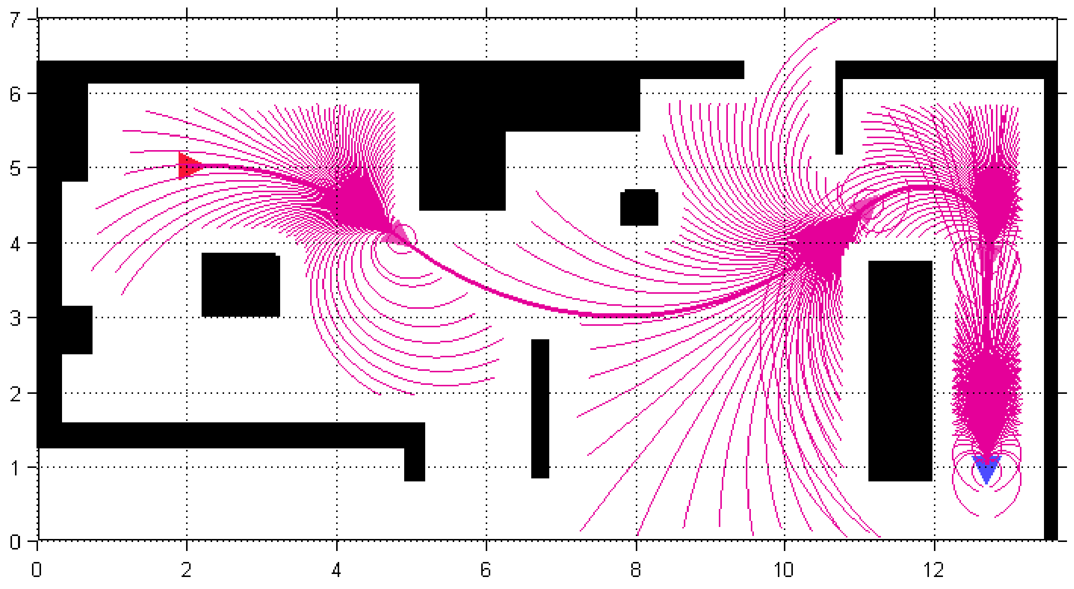


Figure 4.8: Composition of funnels.

Visualization of implicit paths over the domain of attraction of each Lyapunov function (so-called funnels) attached to each node in the graph. These represents the nominal paths the vehicle will take when deviated from the original path.

Fig. 4.8 shows implicit paths over the domain of attraction of each Lyapunov function along the nodes in the path, constructed by tracing the stabilizing vector field attached to each node. Note that at some pose z within the domain of attraction of a node v, the cost-to-go to the final destination is simply $Cost(g) - Cost(v) + Dist(z, v)$ where g is the goal node at the destination pose.

Compared to the original RRT* equipped with Euclidean distance, the proposed algorithm is much more efficient due to proper identification of the nearest and nearby neighbors. We observed up to three times speedup, in agreement with the results reported in *Karaman and Frazzoli* (2013).[5]

4.4.2 Discussion

Having a good measure for the distance between two configurations is very important for planning an optimal path. However, to the best of our knowledge, there is no general consensus of what a 'good' measure is. We propose that to be a good measure, this (distance) function must reflect the true cost-to-go between configurations, and respect the constraints of the system. It needs to be positive definite; it should always be possible to be decreased with some control input, so that it does not create local minima and can be used to generate exact steering. That is, it needs to be qualified as a control-Lyapunov function. We note that it should be possible to extend this RRT* to any system where a control-Lyapunov distance function is available.

In this Chapter, we have presented a non-holonomic distance function for unicycle-type vehicles, and used this distance function to extend the optimal path planner RRT* to handle non-holonomic constraints for unicycle-type vehicles. The critical feature of our proposed distance function is that it is also a control-Lyapunov function, so it better represents the true cost-to-go between configurations and properly reflects the constraints of the system. By using the provably stable control laws and the closed-form distance functions that properly reflect the constraints, our algorithm finds smooth and precise paths that exactly reaches the goal for unicycle-type vehicles, and provides stabilizing vector field and the cost-to-go to the final destination around the planned path by composition of local control-Lyapunov functions. We also directly build on this and develop our stochastic model predictive control for graceful navigation in dynamic and uncertain environment, which is presented in Chapter V.

[5]Also, we note that it should be straightforward to implement a branch-and-bound (*Karaman et al.*, 2011) technique, and the popular k-d tree structure (*Bentley*, 1975) for further improvements in efficiency. For k-d tree implementation, we recommend using (x, y) locations for partitioning, so that it can be used to pre-select a set of candidates for nearby neighbor search. This removes complications that can arise from the use of directed distance.

CHAPTER V

Motion Planning and Control in Dynamic and Uncertain Environments

[1]

In this chapter, we define our stochastic model predictive control for safe, comfortable, and customizable mobile robot navigation in dynamic and uncertain environment. The problem of robot navigation in dynamic and uncertain environment is formulated as online, finite-horizon policy and trajectory optimization problem under uncertainty. With our formulation, the planning and the control are closely integrated, which allows direct optimization of a performance measure.

This depends on several important technical contributions. We define our expected cost (Chapter V) so that we can directly incorporate the time-varying uncertain constraints and the probability of violating those constraints into the cost function and create a smooth cost surface that is easy to optimize over. The dimensionality of this problem is critically reduced with our policy parameterization based on our Lyapunov-stable feedback control law (Chapter III.) The stability of this stochastic model predictive control depends on our choice of the non-holonomic distance function (Chapter IV) as the cost-to-go. Having an accurate model of the robot (Chapter A), and the ability to predict future motion of other dynamic objects in the environment, are also important.

5.1 Introduction

A mobile robot navigating in a dynamic and uncertain environment must be able to handle complex, time-varying constraints presented by the environment and quickly generate

[1]The material in this chapter is not yet published; The basic concept was first introduced in *Park et al.* (2012a) as model predictive equilibrium point control (MPEPC). Here we present a significantly improved formulation of the original idea with better cost definitions.

high-quality trajectory that reaches the goal. It must be sufficiently fast, move smoothly and safely (*Gulati and Kuipers*, 2008) around static and dynamic objects, while respecting the preferences of the user.

Model Predictive Control (MPC) is a receding-horizon control algorithm which optimizes the performance of the constrained systems on-line. (See *Rawlings* (2000), *Mayne et al.* (2000), and *Lee* (2011) for excellent reviews.) MPC can be a very useful tool for mobile robot navigation in dynamic environments due to its dynamic replanning framework, its ability to handle complex time-varying constraints, and its flexible formulation which allows the user to tune the behavior of the system to desired performance; it can also incorporate sophisticated dynamics and interaction models of the robot and other agents in the environment. Thus the MPC and MPC-like dynamic replanning frameworks are becoming more popular in robotics community, (e.g., *Green and Kelly* (2007); *Howard et al.* (2009); *Dolgov et al.* (2010); *Knepper and Mason* (2012), and many others.) However, there are several challenges that need to be addressed to make this framework readily applicable to real systems in dynamic and uncertain environments.

MPC is a mature field with decades of excellent progress, but in general it is still not clear how uncertainties can be handled elegantly in MPC framework (*Lee*, 2011). In robot motion planning, a popular approach in robot motion planning is to set a hard threshold on acceptable probability of collision e.g. (*Du Toit and Burdick*, 2012; *Aoude et al.*, 2013). (This is so-called chance constraint (*Charnes and Cooper*, 1959; *Luders et al.*, 2010).) In optimization problems with such hard constraints, however, solutions typically form at the constraint boundary which can make the solution brittle under noise. Also, to make the robot react adaptively to varying uncertainties in the environment the uncertainties needs to be handled directly in the cost function. To this end, another popular approach is to directly minimize the probability of collision while trying to reach the goal (e.g. (*Trautman et al.*, 2013)), but with those methods it is not clear how to customize robot behavior or to promote higher quality in robot motion such as speed or smoothness.

For high-quality motion generation, motion planning is often formulated as a trajectory optimization problem with a cost function that embodies the desired quality of motion. The cost function is typically constructed as a constant-weighted sum of (often conflicting) sub-objectives (e.g. speed vs. comfort) (*Howard et al.* (2009); *Ogren and Leonard* (2005); *Droge and Egerstedt* (2011); *Gulati* (2011), and many other.) With this approach, however, finding the right weight quickly becomes the real problem, as ill-chosen weights can lead to local minima, non-smooth cost surface, and ultimately to undesirable robot behavior. In general, it is difficult to find a set of constant weights that works well across a wide range of situations.

Here, we present a stochastic MPC which solves optimal motion planning and control problem in dynamic and uncertain environments. We compute the probability of constraint violation as a function of robot and object motion uncertainties in the environment, and directly incorporate this probability into our cost definition. The probability is used to generate time-varying weights that automatically balances the progress toward the goal, the cost of action, and the cost of collision, to compute the overall expected cost.

Our formulation presented in this chapter allows us to optimize the robot behavior by maximizing the expected progress toward the goal, considering the probability of collision and the quality of motion. In the next chapter (Chapter VI), we demonstrate that our algorithm readily generates safe, fast, and smooth motion across wide range of real and challenging situations by a physical robot.

5.2 Optimal Motion Planning in Dynamic and Uncertain Environment

In this section, we review the standard MPC formulation and formally define our motion planning problem.

5.2.1 Standard Discrete-time MPC

Consider the following time-invariant dynamical system in discrete-time,

$$q_{i+1} = f(q_i, u_i), \quad q_k = \hat{q}(t_k)$$

where $q_i \equiv q(t_i) \in \mathbb{Q}$ denote the state (pose, velocity and acceleration), and $u_i \equiv u(t_i) \in \mathbb{U}$ denote the control at time t_i [2] , and q_k is the initial state set equal to the measured value $\hat{q}(t_k)$. For our robot, let $x_i \equiv x(t_i)$ denote the pose (position and orientation), which defines its configuration space.

MPC solves the following finite horizon constrained optimization problem at each time

[2]To be precise, t_k is the so-called sampling time which marks the beginning of the k-th discrete-time interval. It is assumed that the state is measured right before the sampling time, at t_k^-, and the control is applied right after, at t_k^+, and held constant until the end of the interval, to t_{k+1}^-.

step k:

$$\underset{u_{[k\,:\,k+N-1]}}{\text{minimize}} \quad J(k, q_{[k\,:\,k+N]}, u_{[k\,:\,k+N-1]}, N) \tag{5.1}$$

$$\text{subject to} \quad q_{i+1} = f(q_i, u_i), \quad q_k = \hat{q}(t_k) \tag{5.2}$$

$$u_i \in U \subseteq \mathbb{U} \tag{5.3}$$

$$q_i \in Q \subseteq \mathbb{Q} \tag{5.4}$$

where $k + N$ is the (receding) time horizon, U is the constraint on input, Q is the constraint on state, and $J(k, q_{[k\,:\,k+N]}, u_{[k\,:\,k+N-1]}, N)$ is the cost function. This cost function is a scalar function that maps a trajectory of states $q_{[k\,:\,k+N]}$ and inputs $u_{[k\,:\,k+N-1]}$ that obey (5.2) to a cost. In standard formulation, it is usually written as

$$J(\cdot) = \sum_{i=k}^{k+N-1} L(q_i, u_i) + M(q_{k+N}) \tag{5.5}$$

where $L(q_i, u_i) \geq 0$ is the *stage cost* computed along the trajectory, and $M(q_{k+N})$ is the *terminal cost* at the *terminal state* q_{k+N}. This terminal cost is often formulated as a cost-to-go from the terminal state to the eventual goal state, and a proper terminal cost is essential for the stability of MPC.

MPC is very powerful due to its ability to handle complex constraints, and MPCs with this standard formulation (5.1)-(5.5) have been extremely successful in practice, and its properties are well understood (*Rawlings*, 2000; *Mayne et al.*, 2000). However, with this formulation it is not clear how uncertainties can be handled (*Lee*, 2011), which is important in dynamic and uncertain environments.

5.2.2 Stochastic MPC in Dynamic and Uncertain Environments

Now we formulate the stochastic optimization problem by incorporating estimated future uncertainties. Note that the solution $u^*_{[k\,:\,k_N-1]}$ to the optimization problem (5.1) does not change when we subtract a constant value $M(q_k)$ from the cost, so we can replace $J(\cdot)$ with $\tilde{J}(\cdot)$, where

$$\tilde{J}(\cdot) = \sum_{i=k}^{k+N-1} L(q_i, u_i) + M(q_{k+N}) - M(q_k). \tag{5.6}$$

This observation lets us incorporate uncertainties, or more precisely, the probability of constraint violation, directly into the cost.

Suppose the constraint on state is time-varying, and write $Q_i \equiv Q(t_i)$. Let

$$p_c(i) \equiv p_c(q_i, Q_i) \tag{5.7}$$

be the *probability of constraint violation* at time t_i, which is a function of the estimated the robot state q_i and the time-varying constraint Q_i. Naturally, specific definition of this function will depend on the system. For our navigation problem, $Q(t_i)$ is the collision constraint and $p_c(i)$ becomes the probability of collision (5.12)-(5.13) at time t_i.

Based on the probability of constraint violation, we can write

$$p_s(i) \equiv p_s(q_{[k\,:\,i]}, Q_{[k\,:\,i]}) = \prod_{l=k}^{i}(1 - p_c(l)) \tag{5.8}$$

as the *probability of safe transition*,[3] which is the probability of successful (safe) transition from the initial state q_k to the state q_i along $q_{[k\,:\,i]}$ without violating the constraints $Q_{[k:i]}$. With (5.8), we extend (5.5) and define our *expected cost* as

$$E[\tilde{J}(\cdot)] \equiv \sum_{i=k}^{k+N-1} (L(i) + p_s(i) \cdot \Delta M(i) + (1 - p_s(i)) \cdot R(i)) \tag{5.9}$$

where $L(i) \equiv L(q_i, u_i)$ is the stage cost, $\Delta M(i) \equiv M(q_{i+1}) - M(q_i)$ is the chance in the terminal cost-to-go over the trajectory (which we use as measure of progress), and $R(i) > 0$ is the strictly positive cost of constraint violation. The specific definitions of these terms will depend on the system. Note that (5.9) reduces to (5.6) if $p_s(i) = 1$ for all i, that is, if the probability of constraint violation is zero everywhere.

Using (5.9), we write our *optimal motion planning problem in dynamic and uncertain environments* as a stochastic constrained optimization problem

$$\begin{aligned} \underset{u_{[k\,:\,k+N-1]}}{\text{minimize}} \quad & E[\tilde{J}(k, q_{[k\,:\,k+N]}, u_{[k\,:\,k+N-1]}, N)] \\ \text{subject to} \quad & q_{i+1} = f(q_i, u_i), \quad q_k = \hat{q}(t_k) \\ & u_i \in U \end{aligned} \tag{5.10}$$

where the state constraint (5.4) and motion uncertainties are incorporated into the expected cost $E[\tilde{J}(\cdot)]$. Formulation of (5.8) and (5.9) are important contributions of our work.

Furthermore, we utilize control parameterization developed in *Park et al.* (2012a) via feedback control law (*Park and Kuipers*, 2011), for dimensionality reduction and improved

[3]Or the *survivability*, as used in (*Park et al.*, 2012a)

prediction accuracy. Namely, rather than optimizing the control trajectory $u_{[k\,:\,k+N-1]}$ directly, we find optimal parameters of the underlying feedback controller that will lead to the minimum cost trajectory. That is, given a state feedback $u_i = g(q_i, \zeta)$, where ζ denotes the parameters, we have

$$\begin{aligned} \underset{\zeta}{\text{minimize}} \quad & E[\tilde{J}(k, q_{[k\,:\,k+N]}, u_{[k\,:\,k+N-1]}, N)] \\ \text{subject to} \quad & q_{i+1} = f(q_i, u_i), \quad q_k = \hat{q}(t_k) \\ & u_i = g(q_i, \zeta) \end{aligned} \tag{5.11}$$

as our optimization problem, which now is in unconstrained form.

Note that in (5.9), $p_s(i)$ acts as time-varying weight that determines the trade-off between possible constraint violation and the progress toward the goal. This weight is computed on-line as a function of estimated states of the robot and other object. The progress term $\Delta M(i)$ is completely turned off when the probability of safe transition $p_s(i)$ is small, or the probability of constraint violation (collision) is high, so that the solver can focus on avoiding collision. Similarly, the cost of constraint violation $R(i)$ is completely turned off when the probability of collision is zero, which allows the solver to focus on making progress toward the goal and lowering the action cost $L(i)$. That is, with (5.10) the robot is automatically tuned to maximize progress when it is safe, always tries to avoid collision, and never voluntarily move if no trajectory is feasible or is already in collision.

5.3 Robot Navigation via Stochastic MPC

Here, we give an overview of our system and concrete definitions of our cost function. Our MPC has the prediction horizon of $T = 5\,\text{s}$ and replanning rate of $5\,\text{Hz}$. Here we give a brief description of the overall system.

5.3.1 System Overview

For high-quality motion generation via MPC, we need a good model of the robot and other objects in the environment. Our robot is a powered wheelchair which primarily operates in indoor environment. The robot is differentially driven, and is equipped with two on-board lidars (front and back for 360 field of view), odometers, and an IMU. The static environment is mapped in standard occupancy grid via SLAM. See Appendix for more detailed description of the robot model, forward simulation, and low-level control implementation.

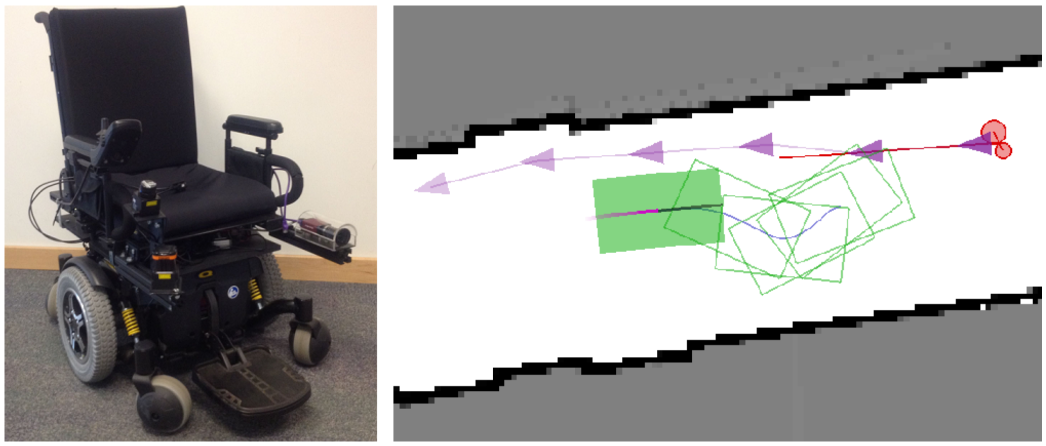


Figure 5.1: The wheelchair robot and an example planner result.

(Best viewed in color) **Left**: The wheelchair robot (0.68m $\times$ 1.1m). **Right**: Example of predicted pedestrian motion (purple) and a planned trajectory (blue) of the robot (green) according to the prediction. Overlaid markers (triangle for pedestrian pose and empty rectangle for robot pose) represent future poses of the robot and the pedestrian, and are placed at 1 sec interval. Our model predicts that the pedestrian will make small corrections in its heading when possible, to avoid collision with the current robot pose (green rectangle) and the walls.

To track dynamic objects, we first identify portion of the laser scans that are not explained by the current static map (*Modayil and Kuipers*, 2008), and those dynamic rays are clustered to form dynamic objects. When two leg candidates are identified, they form a pedestrian object. The validated clusters are tracked via Kalman filters for position and linear velocity. Future trajectories of the tracked dynamic objects (including pedestrians) are predicted via individual MPCs, assuming (i) each dynamic object will maintain speed, but (ii) tries to avoid collision with walls or the current robot pose (at t_k) by slightly modifying the heading at each time step (Fig. 5.1). This simple agent-robot and agent-environment interaction model results in significantly better prediction than the usual constant-velocity models.

Given the SLAM-generated map of the environment and the estimated trajectory of other dynamic objects in the environment, robot trajectory candidates are evaluated for the probability of collision and the expected cost as described in the following subsections.

5.3.2 Policy Parameterization

We use pose-stabilizing kinematic control law (3.13)-(3.17) described in Chapter III as our control policy. The policy, and the resulting trajectories, are parameterized by the motion target $\zeta = (r, \phi, \delta, v_{\text{max}})^T$, which is a temporary target pose in the neighborhood of

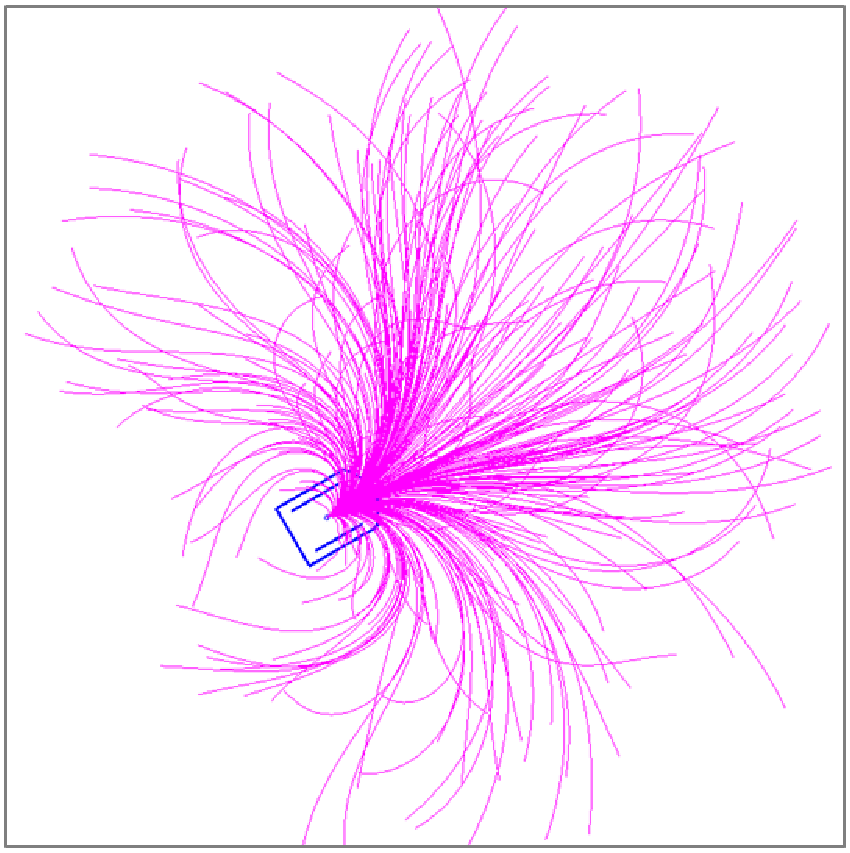


Figure 5.2: Trajectories sampled from the parameterized policy

Randomly selected 300 samples from the continuous space of smooth and realizable trajectories, parameterized by the four dimensional vector $z_* = (r, \phi, \delta, v_{\max})^T$ over a time horizon $[0, T]$. The 4-D vector represents a local target (so-called carrot) and the controller gain. The trajectories are generated from dynamically simulated vehicle (with non-holonomic constraints and actuator saturation) under the stabilizing kinematic controller.

the robot (so-called carrot) and a gain value. This parameterization via the smooth control law allows the planning algorithms to focus on the space of trajectories that are smooth and realizable (Fig. 5.2.)

5.3.3 The Probability of Collision

The most important constraint for robot navigation is to avoid collision. Given predicted trajectories of the robot and other object over $[t_k, t_{k+N}]$, we model the probability of collision of the robot with j-th object in the environment across a small time interval $[t_{i-1}, t_i]$ as a bell-shaped function:

$$p_c^j(d^j(t_i),\, \sigma^j(t_i)) \equiv exp(-d^j(t_i)^2/\sigma^j(t_i)^2) \tag{5.12}$$

where $d^j(t_i)$ is the (numerically determined) minimum distance from any part of the robot body to any part of the j-th object at time t_i, and $\sigma^j(t_i)$ represent the combined motion uncertainty of the robot and the object. Static structures in the environment is considered as a single object in the environment.

For the overall probability of collision to any object in the environment $p_c(i)$ in time interval $[t_i, t_{i+1}]$, we look at the object with the highest probability of collision, so that

$$p_c(i) \equiv \max_j \{p_c^j(d^j(t_{i+1}),\, \sigma^j(t_{i+1}))\} \tag{5.13}$$

then the probability of successful transition (without collision) $p_s(i)$ from the initial state q_k to the state q_{i+1} at the end of i-th segment is

$$p_s(i) = \prod_{l=k}^{i}(1 - p_c(l)), \quad i \geq k \tag{5.14}$$

which is computed recursively through the estimated trajectory. At the initial time interval, $p_s(k) \equiv (1 - p_c(k))$. Note that with (5.14), $p_s \in [0, 1]$ and never increases as time progresses, i.e. $p_s(i) \geq p_s(j),^{\forall}\, i \leq j$.

Having a good approximation of the motion uncertainties is important for safe and effective navigation. To be safe, the measure of uncertainty $\sigma^j(\cdot)$ needs to be larger than true motion uncertainty of the robot. We also note that for passenger vehicles this needs to be larger than *perceived* motion uncertainty of the robot by the user so that the robot keeps comfortable clearance to objects, since the robot may appear reckless otherwise. It must not be too conservative, however, to allow the robot to navigate in crowded and cluttered environments. We use the following approximation which works well in practice:

$$\sigma^j(t_i) = c_0^j + c_t(t_i)\sqrt{c_{\sigma_v} v(t_i)^2 + c_{\sigma_\omega}\omega(t_i)^2} \tag{5.15}$$

where c_0^j is an object-dependent constant, $v(t_i), \omega(t_i)$ are estimated linear and angular velocities of the robot at t_i, and $c_t(\cdot) > 0$ is a coefficient that models the (bounded) expansion of the uncertainty along the estimated trajectory. The nominal values used in our experiments in the next chapter are $c_0^j = 0.02$ for static objects, $c_0^j = 0.08$ for dynamic objects, $c_{\sigma_v} = 0.02$, $c_{\sigma_v} = 0.01$, and $c_t(t_i) = \min((t_i - t_k)/T_s,\, 1)$ with $T_s = 2$.[4]

[4]These parameters were selected so that the uncertainty estimate captures the prediction accuracy of our robot and pedestrian models.

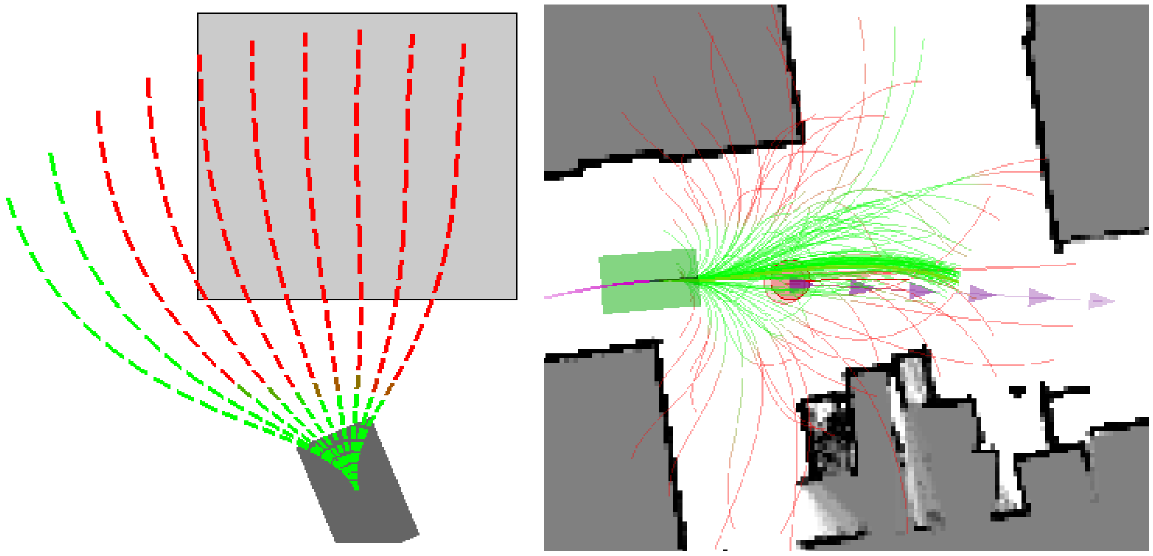


Figure 5.3: Visualization of the probability of successful transition along each trajectory.

(Best viewed in color) **Left**: Cartoon illustration of the cost. With the transition probability p_s, $R(i)$ always pushes the solution away from obstacles, $\Delta M(i)$ pulls it toward the goal. $L(i)$ can be used to modify robot behavior.
Right: Example valuation of the transition probability on a physical robot. Red to green indicates low to high probability of successful transition from the initial state. The probability of collision is computed considering the static structure and the estimated trajectory of the pedestrian (purple) moving in front of the robot (green rectangle).

5.3.4 Expected Cost of a Trajectory

For stage cost $L(i)$ over the trajectory, we have quadratic cost on velocities and accelerations:

$$L(i) = (c_v(v(t_i)^2 + c_\omega \omega(t_i)^2) + c_a(a(t_i)^2 + \gamma(t_i)^2)) \cdot h_i \tag{5.16}$$

where c_v is the weight for the overall velocity, c_ω is the weight angular velocity and curvature, c_a is a (small) weight on linear acceleration $a(t_i)$ and angular acceleration $\gamma(t_i)$, and $h_i \equiv t_{i+1} - t_i$ is the time interval between samples. This choice of the stage cost (5.16) allows us to easily change the qualitative behavior of the robot by changing the weights. Namely, the choice of c_v and c_ω determines the preferred speed of the robot, controlling apparent aggressiveness of robot motion. In our experiment, nominal value of the weights are $c_v = 0.4$, $c_\omega = 0.2$ and $c_a = 0.05$. We have constant $h_i = 0.2$ for all i.

For the collision cost $R(i)$ over the trajectory, we have

$$R(i) = r_0 + r_v(|v(t_i)| + |\omega(t_i)|) \cdot h_i \tag{5.17}$$

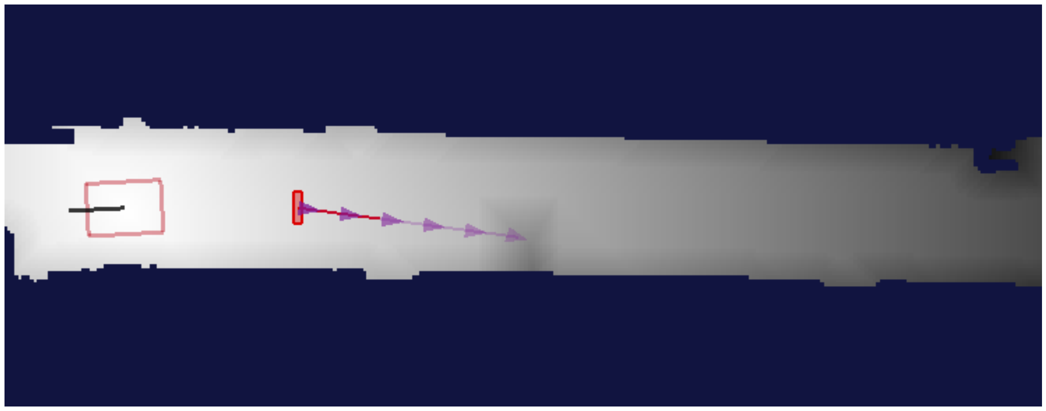


Figure 5.4: Navigation Function example.

A typical navigation function computed in navigable space, over which we measure the progress toward the goal. The goal pose depicted as an empty rectangle with a protruding line segment on the left. Note that we add occupancy cost around the estimated object position at the end of the prediction horizon, as well as around static obstacles in the environment. See text for details.

which helps the optimizer to quickly move away from trajectories that involves collision. For constants we have $r_0 = 0.5$, $r_v = 0.5$, and $h_i = 0.2$ for all i.

For the progress to the goal $\Delta M(i)$, we use the non-holonomic distance developed in Chapter IV when the robot is in the neighborhood of the goal pose, or the grid-based navigation function $NF(\cdot)$ and its gradient (*Konolige*, 2000) computed over navigable space. The navigation function essentially encode the collision-free distance-to-go to the destination and its gradient represent the optimal heading from any point in the navigable space in static environment.

As suggested in *Konolige* (2000), we also add small occupancy cost around static objects (within 0.4m, slightly larger than half width of the robot.) More importantly, we also add small occupancy cost around the estimated position of dynamic object at the end of prediction horizon to better guide the robot (See Fig. 5.4).

The expected progress over the trajectory is written as

$$\sum_{i=k}^{k+N-1} p_s(i)(NF(t_{i+1}) - NF(t_i)) + p_s(k+N)(\delta_e(t_{k+N}) - \delta_e(t_k)) \tag{5.18}$$

where $NF(t_{i+1}) - NF(t_i)$ measures piecewise progress over the navigation function, and $\delta_e(t_i) \equiv c_\theta|\theta(t_i) - \theta^*(t_i)|$ is the weighted ($c_\theta = 0.5$) angular error in robot heading with respect to the vector field. The vector field $\theta^*(t_i)$ is defined by the gradient of the navigation function or by the non-holonomic distance function (as described in Chapter IV).[5] Both $NF(\cdot)$ and $\delta_e(\cdot)$ are required for convergence. Finally, our expected cost of the trajectory to be minimized is the sum of (5.16), (5.17), and (5.18).

5.3.5 Implementation

With our problem formulation the computational cost for the numerical optimization is small, achieving real-time performance. A typical optimization cycle takes $\simeq 60$ ms with $\simeq 400$ trajectory evaluations with our C++ implementation on a 2.66-GHz laptop, using off-the-shelf optimization packages (NLOPT, *Johnson*.) We implement two-stage optimization, with randomized coarse initial search to find an approximate solution, and local gradient based search for final refinement of the solution.

As the planning, mapping, and tracking all take non-zero amount of time, accounting for the computational delay becomes important for accurate motion planning. We explicitly take into account the timing of the sensor information used to estimate the state of the robot and dynamic objects, and use respective models to pre-synchronize their state to the end of the current planning cycle and then compute the optimal control from the beginning of the next planning cycle to the planning horizon. The computed control is executed exactly at the beginning of the next planning cycle to enforce strictly constant update rate.

5.3.6 Remarks

Our formulation presented in this chapter allows us to optimize the robot behavior by maximizing the expected progress toward the goal, considering the probability of collision and the quality of motion. In the next chapter (Chapter VI), we demonstrate that our algorithm readily generates safe, fast, and smooth motion across wide range of real and challenging situations by a physical robot.

[5] We only promote alignment at the end pose rather than along the entire trajectory to give the robot more flexibility. The angular term is essential for stability of the goal, as it properly motivates the robot to turn in place in tight corridors and at the origin (goal). This also makes the cost-to-go qualify as CLF for vehicle kinematics. See *Mayne et al.* (2000) and *Jadbabaie et al.* (2001) for stability analysis of the origin under MPC.

CHAPTER VI

Evaluations

In this chapter, we evaluate our MPEPC algorithm applied to a physical robot. The evaluation is done both in dynamically simulated environments generated from real data traces and in physical environments with a real robot. In Section 6.1, we focus on showing that the algorithm can react well to dynamic objects in the environment, and show that the robot behavior (apparent aggressiveness) is easily customizable by applying the algorithm with varying level of action costs to a replayed dataset recorded from the physical robot. In Section 6.2, we apply the algorithm to the task of moving alongside (pacing) and behind (following) a person, and show that the algorithm adapts well to different preferences of the user as well as to dynamic objects in the environment. In Section 6.3, we present the results from experiment on a physical robot. The main experiment in Section 6.3 contains a 7 minutes of uninterrupted robot navigation in a typical corridor environment in the presence of pedestrians (Fig. 6.13 - 6.19). The robot navigates successfully without any collision (Fig. 6.13). The unmodified algorithm exhibits a wide range of reasonable and apparently intelligent behaviors such as passing, moving around, following behind, and waiting for a person to pass in a narrow corridor. Snapshots of individual behaviors with visualizations of the optimization process are shown in Fig. 6.14 - 6.19. We also show an example run (without collision) in a crowded situation (Fig. 6.20 - 6.22.)

As can be seen in the examples, the algorithm can generate safe and smooth trajectories across a range of situations exhibiting very reasonable behavior, and the robot motion is easily customizable. The overall algorithm is very efficient and achieves real-time performance thanks to the low-dimensional parameterization of the trajectory space, and does not require any post-processing to compute the control inputs as the feedback control policy is directly being optimized.

We note that in realistic settings, for an agent with limited actuation capability (non-holonomic and dynamic constraints) and without perfect knowledge of the environment, it is not possible to guarantee absolute collision avoidance. What can be guaranteed with this algorithm, however, is that the robot will never voluntarily move if collision is inevitable; the expected cost definition motivates the robot to slow down as the probability of collision increases, and forces the robot to stop when the probability is high.

6.1 Robot Navigation in Simulated Environments

[1] Initially, we have tested the algorithm in two typical indoor environments: a tight L-shaped corridor, and an open hall with multiple dynamic objects (pedestrians). In the L-shaped corridor, the proposed algorithm allows the robot to exhibit wide range of reasonable motions in different situations, e.g. to move quickly and smoothly in an empty corridor (Fig. 6.1), and to stop, start and trail a pedestrian in order to progress toward a goal without collision (Fig. 6.2). In the hall environment, we show that the algorithm can deal with multiple dynamic objects, and that changing a weight in the cost definition can result in qualitatively different robot behavior in the same environment (Fig. 6.4 - 6.5).

All sensor data are from actual data traces obtained by the wheelchair robot (Fig. 3.7). The position and velocity of dynamic objects are tracked from traces of laser point clusters, and the planner estimates the motion of the dynamic objects using a constant velocity model over the estimation horizon $[0, T]$. Robot motion is dynamically simulated within the environments generated from the sensor data.

Fig. 6.1 shows a sequence of time-stamped snapshots of the robot motion as it makes a right turn into a narrow corridor (Top), the trajectories sampled by the planner (Bottom, gray), the time-optimal plans selected at each planning cycle (blue), and the actual path taken (red). The robot moves swiftly through an empty corridor along the gradient of the navigation function near the allowed maximum speed (1.2 m/s) (Fig. 6.3, Top) toward the goal pose. Once the robot moves within 2 m of the goal pose the robot switches to a docking mode and fixes the motion target at the goal. In this example, the robot safely converges to the goal pose in about 18 s.

In Fig. 6.2, the robot runs with the same weights in the cost function but with a slow moving (~ 0.5 m/s) pedestrian in the map. The tracked location (red circle) and the estimated trajectory (red line) of the dynamic object is shown. In order to progress toward the goal without collision, the robot effectively trails the pedestrian, stopping and restarting as needed. A comparison of the robot speed and the pedestrian speed is shown in Fig. 6.3, Bottom.

In Fig. 6.4 - 6.5, robot motion in an open hall with multiple dynamic objects is shown. For these examples, the bound on linear velocity (and the overall gain) is increased to

[1]The material in this section is a revised presentation of the results in *Park et al.* (2012a,b).

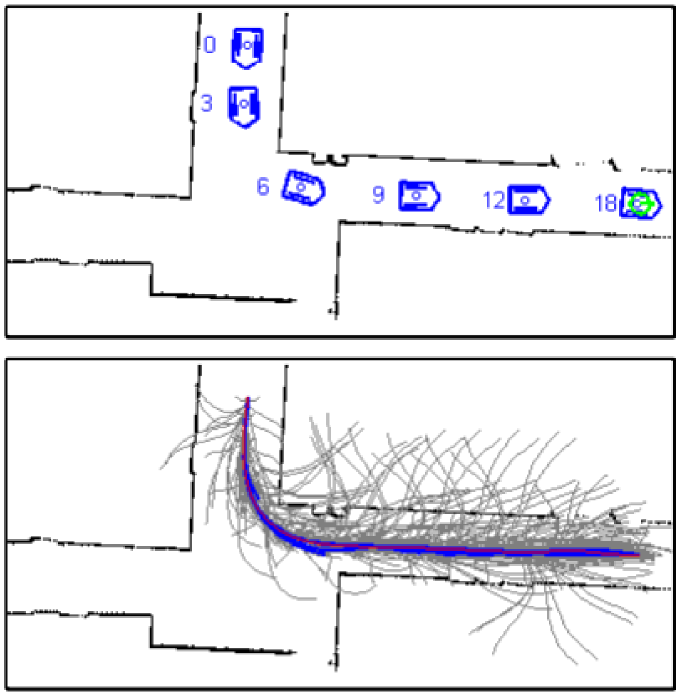


Figure 6.1: Robot navigation example in L-corridor.

(Best viewed in color) **Top**: Time-stamped snapshots of the motion of the robot (0.68×1.1 m), in a L-shaped corridor with a small opening (2 m wide, smallest constriction at the beginning 1.62 m). The robot moves quickly and smoothly in the free corridor. See Fig. 6.3 for velocities and accelerations along the trajectory.
Bottom: Trajectories sampled by the planner (gray), time-optimal plans at each planning cycle (blue), and the actual path taken (red).

$v_{max} \leq 1.9$ m/s. In Fig. 6.4, the weights for the linear and angular velocity cost is set higher ($c_v = 0.4$ and $c_\omega = 0.2$) so the robot prefers to move slower, moving behind the pedestrian C as it passes and then speeding up. In Fig. 6.5, with a small weight on the linear velocity cost ($c_v = 0.04$ and $c_\omega = 0.02$) the robot is more aggressive and does not slow down, cutting in front of the slow-moving pedestrian C. The trajectories sampled by the planner (gray), time-optimal plans at each planning cycle (blue), and the actual path taken (red) are shown on the right.

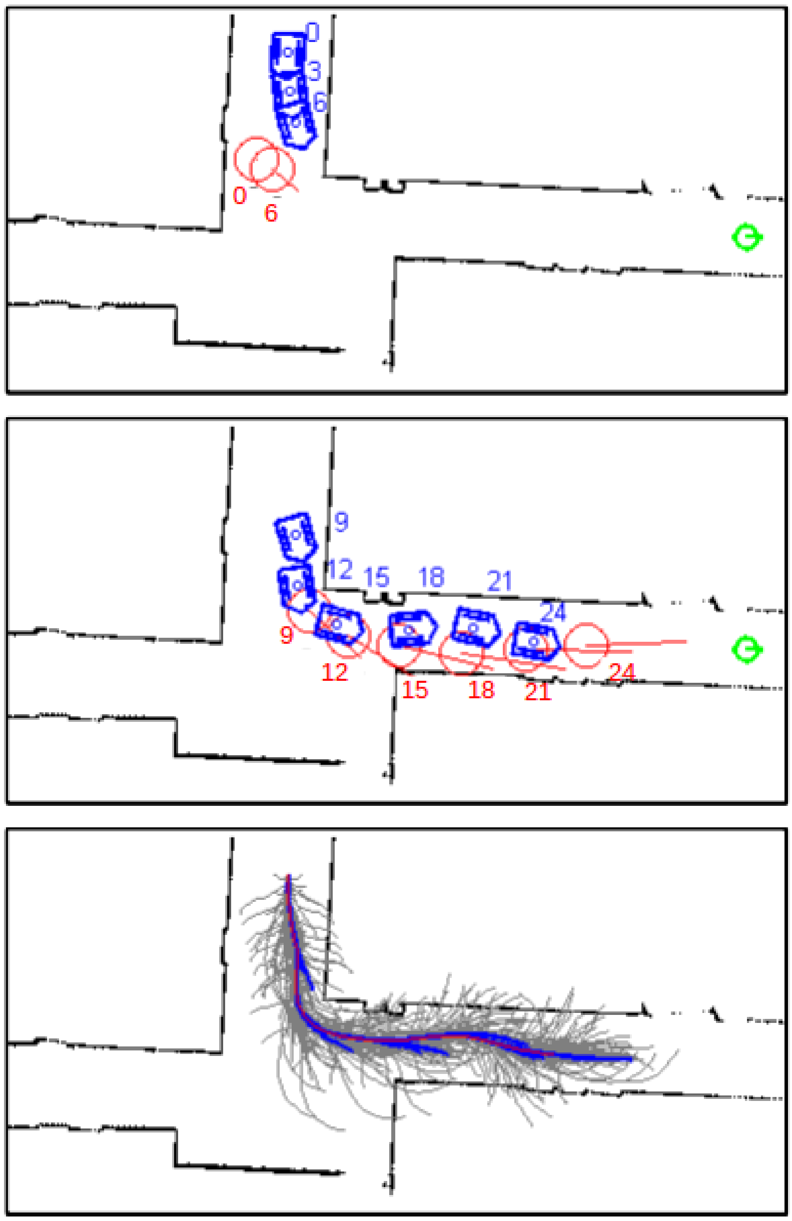


Figure 6.2: Robot navigation example in L-corridor, with a pedestrian.

(Best viewed in color) **Top**: The robot motion in 0 - 6 s. As a pedestrian (red circle) begins to move, the robot stops briefly to avoid collision (at $t \sim 6$ s). Velocities and accelerations shown in Fig. 6.3. Estimated trajectory of the dynamic object is shown as red lines.
Middle: As the pedestrian moves into the corridor, the robot starts trailing the pedestrian at about the same average speed, progressing toward the goal while avoiding collision.
Bottom: Trajectories sampled by the planner (gray), time-optimal plans at each planning cycle (blue), and the actual path taken (red).

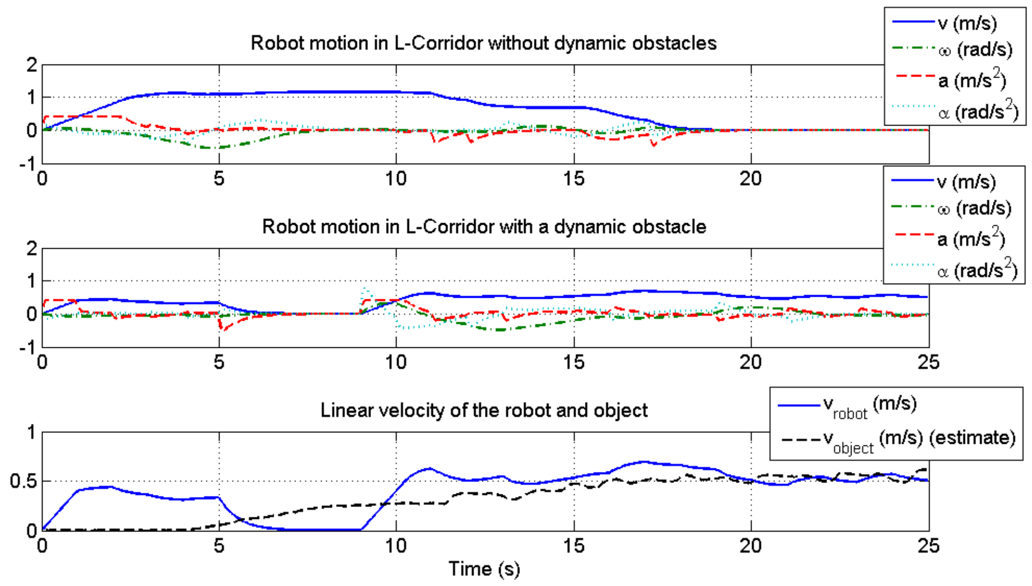


Figure 6.3: Robot velocity and acceleration profiles for L-corridor example.

(Best viewed in color) **Top**: Velocity and acceleration profiles for the robot in the free corridor. The vehicle is given a very small weight on linear velocity cost and moves close to the maximum allowed speed.
Middle: In a corridor with a pedestrian, the robot is able to stop, start again and trail the pedestrian with the proposed algorithm, without changing any parameters.
Bottom: Linear velocity profile of the robot and the pedestrian (dynamic object). The robot stops briefly at $t \sim 6$ s as the pedestrian blocks the way, and trails the pedestrian nearly at the same speed as it moves.

6.2 Person Pacing and Following in Simulated Environments

[2] In this Section, we introduce a navigation algorithm for *person pacing*, which refers to the capability to walk next to another person at desired distance and orientation (*Miller*, 1996). Person pacing is a very important skill for a service robot, especially for autonomous wheelchair applications where a passenger commonly wants to accompany a person side by side. Note that for such passenger vehicles, there are very important additional requirements such as safety, comfort, and preference of the passenger that cannot be overlooked.

6.2.1 Introduction

For motion control, existing literatures on the topic tend to focus only on *person following*, where the objective is simplified to maintain relative distance without considering orientation. There are published methods based on P- or PID-control and its variants, (*Topp*

[2]The material in this section is a revised presentation of the results in *Park and Kuipers* (2013).

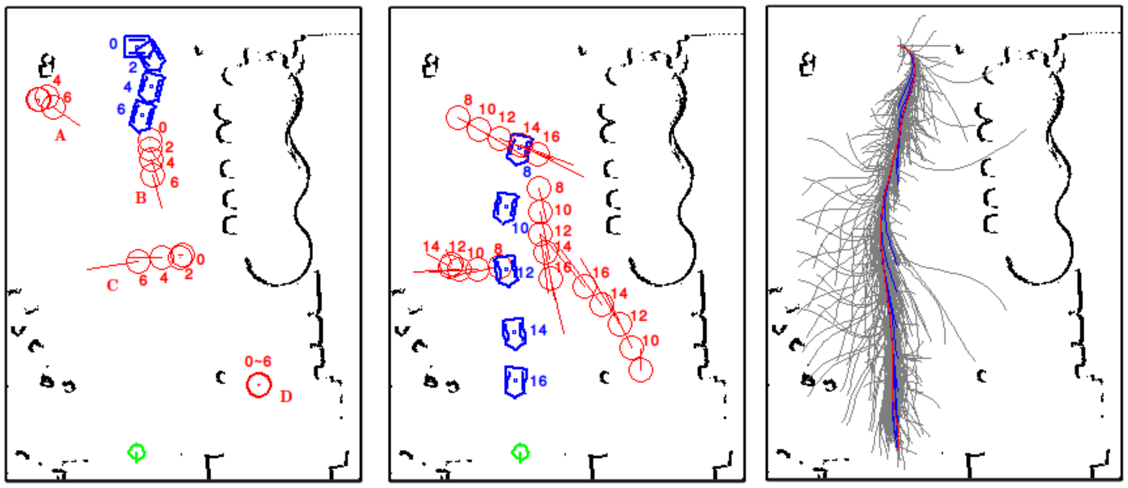


Figure 6.4: Robot navigation example in open hall (passive)

(Best viewed in color) In an open hall (13.5 $\times$ 18 m) with multiple pedestrians (A-D, red circles), the robot (blue) estimates the trajectories of dynamic objects over the planning horizon (red lines) and navigates toward the goal (green circle) without collision. Compared to Fig. 6.5 the robot moves slower due to larger weights on the action cost ($c_v = 0.4$ and $c_\omega = 0.2$). **Left and Center**: Time-stamped snapshots of the robot and pedestrian. **Right**: Trajectories sampled by the planner (gray), time-optimal plans selected at each planning cycle (blue) and the actual path taken (red).

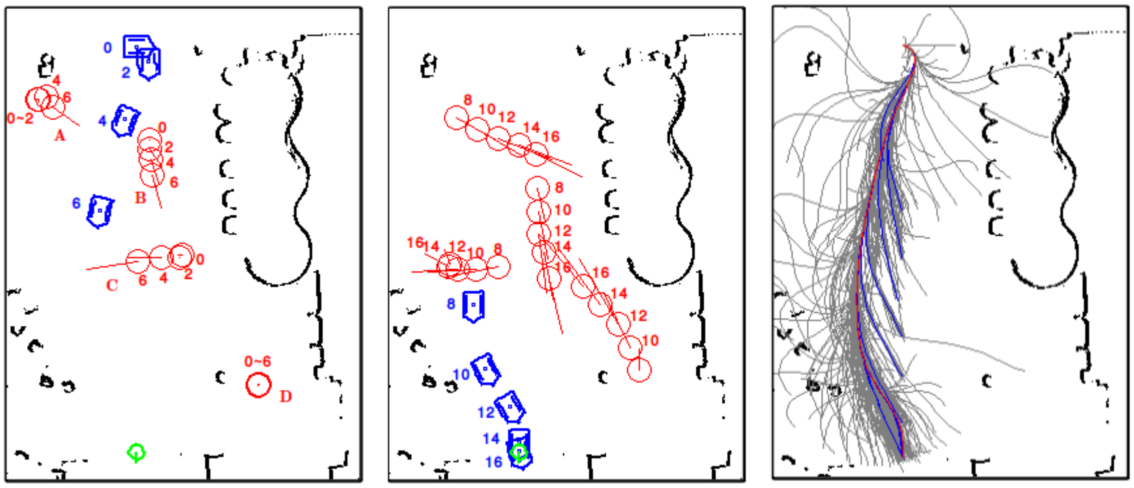


Figure 6.5: Robot navigation example in open hall (aggressive)

(Best viewed in color) Robot motion in the same environment as Fig. 6.4, but with low weights on the action cost ($c_v = 0.04$ and $c_\omega = 0.02$). The robot moves more aggressively and cuts in front of the dynamic object marked C, which it passed behind in Fig. 6.4. **Left and Center**: Time-stamped snapshots of the robot and pedestrian. **Right**: Trajectories sampled by the planner (gray), time-optimal plans selected at each planning cycle (blue) and the actual path taken (red).

and Christensen, 2005; *Gockley et al.*, 2007; *Ma et al.*, 2008; *Germa et al.*, 2009; *Hu et al.*, 2009), simple distance and distance-and-velocity based control laws, (*Satake and Miura*, 2009; *Takemura et al.*, 2009), a finite state machine (*Lam et al.*, 2011), and on-line probabilistic roadmap (*Frintrop et al.*, 2010; *Hoeller et al.*, 2007), but the resulting motion from those methods tends to be slow, jerky, or both. Notably, there is a classical method based on velocity obstacle for person pacing (*Prassler et al.*, 2002), and more recent work (*Hemachandra et al.*, 2011) that uses a variant of pure pursuit for person following with good performance, but smoothness and comfort of robot motion are largely neglected in existing literature. Also, as several studies (*Walters et al.*, 2005; *Gockley and Mataric*, 2006) have found, the appropriate distance may vary according to individuals (*Gockley et al.*, 2007), but with existing methods typically use a fixed value for desired clearance and it is not clear how to adapt to user-specified preferences in distance and orientation.

Here we develop a versatile motion planning algorithm that is able to generate behaviors that are appropriate across a wide range of situations, while respecting user-specified distance and orientation preferences. The algorithm considers the robot's and the person's current configuration, observed locations and velocities of pedestrians, and static structures in the environment, while moving as smoothly and comfortably as possible.

6.2.2 Cost definition for Person Pacing and Evaluations

Formally, let $\rho(t)$ be the estimated radial distance between the robot and the target person at time t, and $\eta(t) \in (-\pi, \pi]$ denote the estimated relative orientation of the robot as measured from the heading of the target person at time t. We implement a straightforward cost metric $F(\cdot)$ for person pacing (which is in unit of distance),

$$F(t) = |\rho(t) - \rho_d| + c_\eta \cdot \rho_d \cdot |g(\eta(t) - \eta_d)| \tag{6.1}$$

where ρ_d and η_d denote the user-specified desired distance and orientation in the person-frame,[3] c_η is the weight for orientation error, and $g(\cdot)$ is a function that wraps any angle to interval $(-\pi, \pi]$.

If there is no specific preference for the orientation, the problem reduces to person following and $F(t) = |\rho(t) - \rho_d|$. Also, if there are more than two desired orientations, e.g. walking on either side of the person is equally desirable, we replace the orientation error $|g(\eta(t) - \eta_d)|$ with $\min(|g(\eta(t) - \eta_{d1}|, |g(\eta(t) - \eta_{d2})|, ...)$. Example cost surfaces are shown in Fig. 6.6.

[3]The desired orientation is defined with respect to the heading of the target person, and is not a function of robot orientation in reference frame.

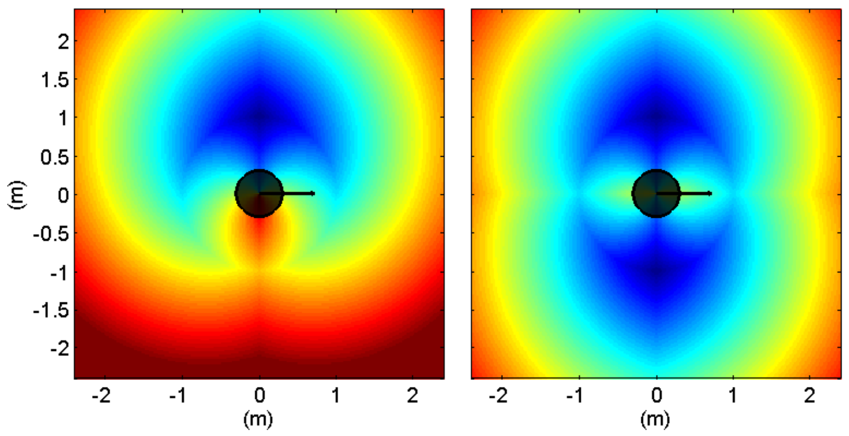


Figure 6.6: Moving cost-to-go definition for person pacing.

(Best Viewed in Color) Example cost surfaces for person pacing which encode the user specified desired distance and orientation. The person to follow is depicted as a circle in the middle, with arrow indicating the person's heading. Red-to-blue color represents high-to-low cost. The cost surface is attached to the estimated pose of the person. See text for details.
Left: With the user specified desired orientation of $\pi/2$ rad (on the left side of the person) and the desired distance of 1 m.
Right: The desired distance is the same at 1 m, but with the preferred orientation of $\pm\pi/2$ rad (on either side of the person).

In Fig. 6.7, the robot starts at some distance away from the target pedestrian and quickly sets itself up on the left side of the person, with desired distance and orientation of $\rho_d = 1.5$ m and $\eta_d = \pi/2$. The mission is relatively easy, and the robot is able to follow the person smoothly near desired distance and orientation (Fig. 6.8, Right). The full set of trajectories explored over the navigation run is shown on Fig. 6.7, Right. Overall, the robot moves smoothly, except a brief slowdown near 14 s (Fig. 6.8) to avoid collision with a pedestrian coming across.

Fig. 6.9 shows the trajectory of the robot in the same environment, but it is now instructed to pace the person on the right, with desired distance and orientation of $\rho_d = 1.5$ m and $\eta_d = -\pi/2$. The mission is more difficult, as the robot has less space to maneuver. The successfully speed up, slow down, change its heading to avoid the pedestrian and return to its desired position. The full set of trajectories that were explored over the navigation run is shown on Fig. 6.7, Right. As can be seen in this example, the robot with the proposed navigation algorithm is able to temporarily move away from the user-specified distance and orientation preferences to avoid collision with other obstacles (Fig. 6.10).

Fig. 6.11 - 6.12 show the most challenging example, where the robot (0.76×1.2 m) has to move in a tight corridor (2 m wide) with multiple pedestrians. In this example no

specific preference in orientation is given, and the problem is reduced to person following with the desired distance of $\rho_d = 1.8\,\mathrm{m}$. The example shown is very difficult as the robot needs to move in confined space and there is a pedestrian quickly moving in from behind. The robot is able to successfully follow the person, speeding up, switching sides, slowing down and veering as needed.

Over the three example runs shown, 10240 trajectory candidates were evaluated in 166 planning cycles, averaging 61.7 trajectory evaluations per optimization, achieving real-time performance.

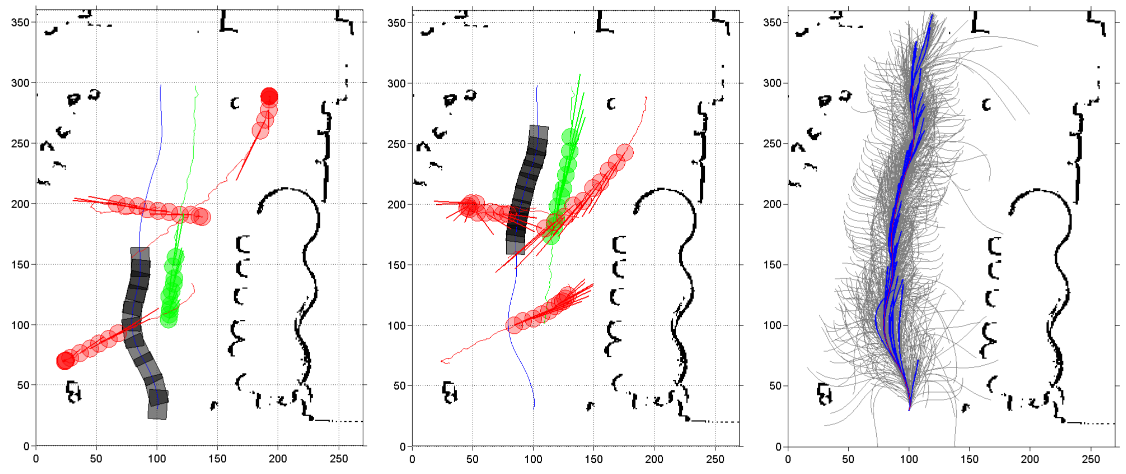


Figure 6.7: Person pacing example in an open hall.

(Best viewed in color) In an open hall ($13.5 \times 18\,\mathrm{m}$) with multiple pedestrians (circles), the robot (blue) estimates the trajectories of dynamic objects over the planning horizon (protruding lines) and paces the target person (green) on the left without collision. This is a relatively easy mission. **Left and Center**: The trajectories overlaid with snapshots of the robot (rectangle) and the pedestrians (circles) at $1\,\mathrm{s}$ interval. The axis are in grid coordinates (5cm per grid). The robot starts from the bottom, trying to pace the person in the middle on the left.

Right: Trajectories sampled by the planner (gray), time-optimal plans selected at each planning cycle (blue) and the actual path taken (red).

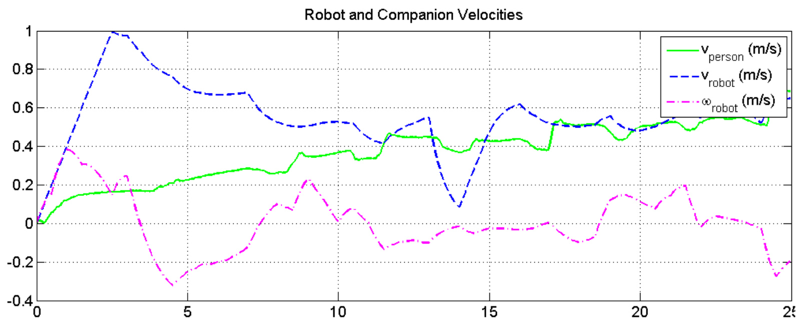


Figure 6.8: Robot and companion velocities in the example shown in Fig. 6.7.

Estimated velocity of the leading pedestrian and the measured speed of the robot. The robot moves smoothly, and shows a brief slowdown near $14\,\mathrm{s}$ to avoid collision with a pedestrian coming across its path.

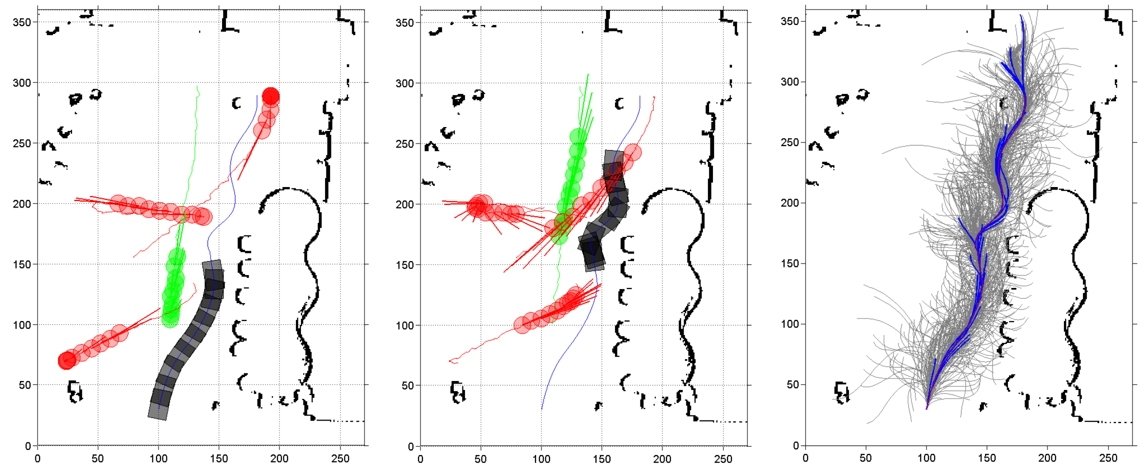


Figure 6.9: Person pacing example in an open hall (more difficult).

(Best viewed in color) In an open hall ($13.5 \times 18\,\mathrm{m}$) with multiple pedestrians (circles), the robot (blue) estimates the trajectories of dynamic objects over the planning horizon (protruding lines) and paces the target person (green) on the right without collision. The robot has to maneuver in a tighter space and avoid an oncoming pedestrian. The successfully speed up, slow down, change its heading to avoid the pedestrian and return to its desired position.
Left and Center: The trajectories overlaid with snapshots of the robot (rectangle) and the pedestrians (circles) at 1 s interval. The axis are in grid coordinates (5cm per grid). The robot starts from the bottom, trying to pace the person in the middle on the right. The robot is able to move away from the user-specified distance and orientation targets to avoid collision with other obstacles.

Right: Trajectories sampled by the planner (gray), time-optimal plans selected at each planning cycle (blue) and the actual path taken (red).

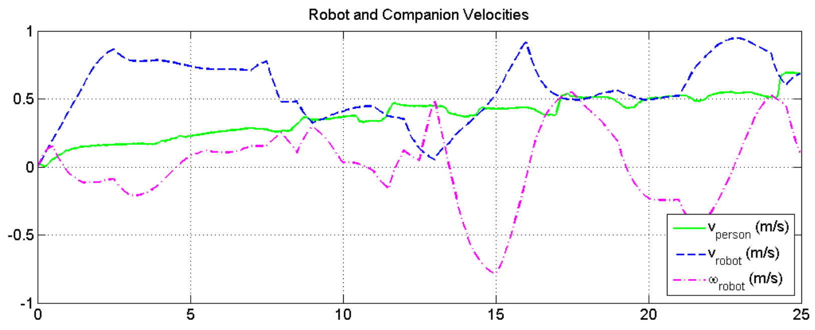


Figure 6.10: Robot and companion velocities in the example shown in Fig. 6.9.
Estimated velocity of the leading pedestrian and the measured speed of the robot.

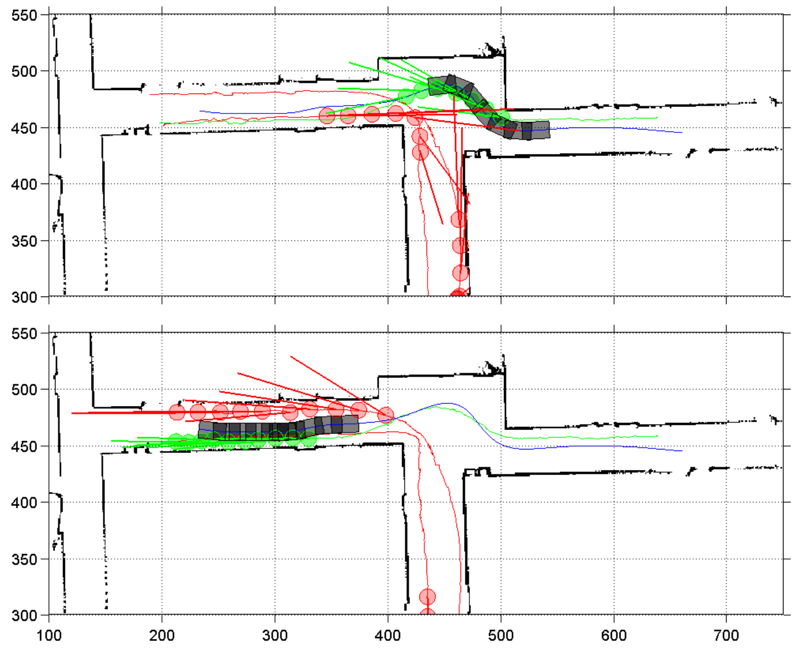


Figure 6.11: Person following example in a narrow corridor.

(Best viewed in color) In a tight corridor environment (32.5×12.5 m, 2 m in width) with multiple pedestrians (circles), the robot (blue) estimates the trajectories of dynamic objects over the planning horizon (protruding lines) and follows the target person (green), without orientation preference. The axis are in grid coordinates (5cm per grid).
Top: The robot follows the person slightly on the left to keep distance from walls, and can move away from estimated trajectory (red lines) of an oncoming pedestrian.
Bottom: After the robot moves into a hallway, the person moves along the left wall and the robot switches to the right side of the person to keep distance from walls. When it detects a quickly approaching pedestrian from behind, it slows down and veers slightly left to avoid collision with the pedestrian and the person the robot is following.

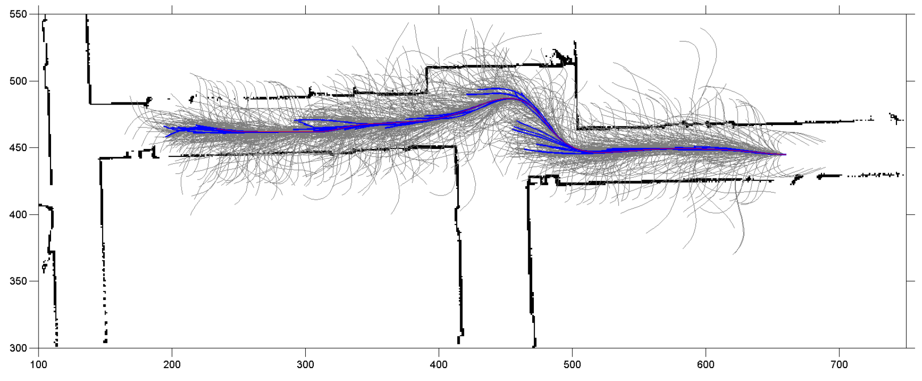


Figure 6.12: Trajectories evaluated by the planner in the example shown in Fig. 6.11.

(Best viewed in color) Densely sampled candidate trajectories in the neighborhood of the robot over the navigation run in Fig. 6.11 (gray), time-optimal plans selected at each planning cycle (blue) and the actual path taken (red).

6.3 Robot Navigation in Real Environments

Here we show results from the experiments carried out in real indoor environments with a physical wheelchair robot (Fig. 3.7). Mapping, tracking, planning, and control was performed on-line and in real time from an on-board laptop. We show that the robot exhibits a series of useful and seemingly intelligent behaviors as a result of direct optimization of the expected cost.

The main experiment in this section contains a 7 minute robot navigation in a typical corridor environment in the presence of pedestrians (Fig. 6.13 - Fig. 6.19.) We show that with our stochastic MPC, the robot navigates successfully without any collision (Fig. 6.13), and exhibits a wide range of reasonable and seemingly intelligent behaviors such as passing, moving around, following behind, and waiting for a person to pass in a narrow corridor. Snapshots of individual behaviors with visualization of the optimization process are shown in Fig. 6.14 - Fig. 6.19. The robot is able to smoothly avoid an oncoming pedestrian (Fig. 6.14), circumvent a standing person in the middle of the way (Fig. 6.15), exhibit apparent following behavior in narrow passage behind a person (Fig. 6.16), safely overtake a person when possible (Fig. 6.17), or decides to walk alongside if the weight on the action cost is high (Fig. 6.18), and can even wait for a pedestrian to clear the way. See captions for details.

Next, we tested the algorithm in more crowded environment (Fig. 6.20-Fig. 6.22). The robot is able to move through crowd without any collision, and always will slow down/stop if the probability of collision is high.

We demonstrate that our method generates graceful (safe, smooth, comfortable, fast, and intuitive) and customizable robot behavior in physical environments with pedestrians in real time, which is difficult to achieve with existing algorithms. Unlike other 'safe' algorithms which rely on a slow moving robot and very conservative estimates, our stochastic MPC can fully utilize the robot's physical capability and can move the vehicle at its top speed when it is safe (and if the user prefers to move fast), but is guaranteed to slow down gracefully and move carefully if the perceived probability of collision is high. This is an important and necessary feature for a motion planning algorithm for a passenger-carrying vehicle. We believe that our work is a significant step toward safe and reliable autonomous navigation that is acceptable to human users.

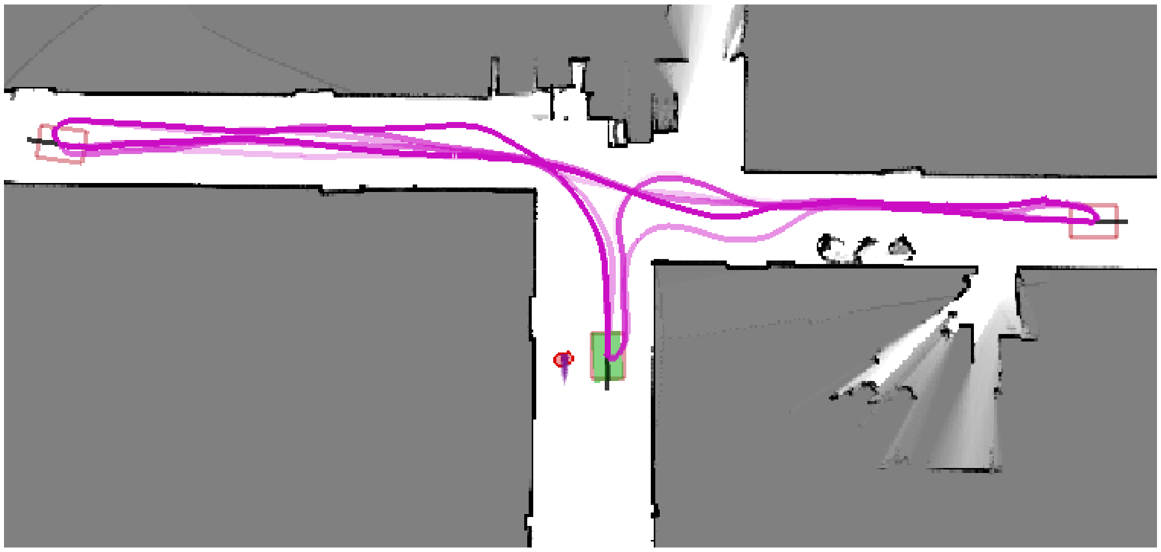


Figure 6.13: Accumulated robot trace from a physical robot in dynamic environment.

The robot (green rectangle) looping through 3 target poses autonomously in 22x15m corridor environment in the presence of pedestrians. Magenta line in this figure represents accumulated 7 minutes of robot trace, where the robot navigated successfully without any collision. The variability of the trace is the result of the robot avoiding pedestrians.

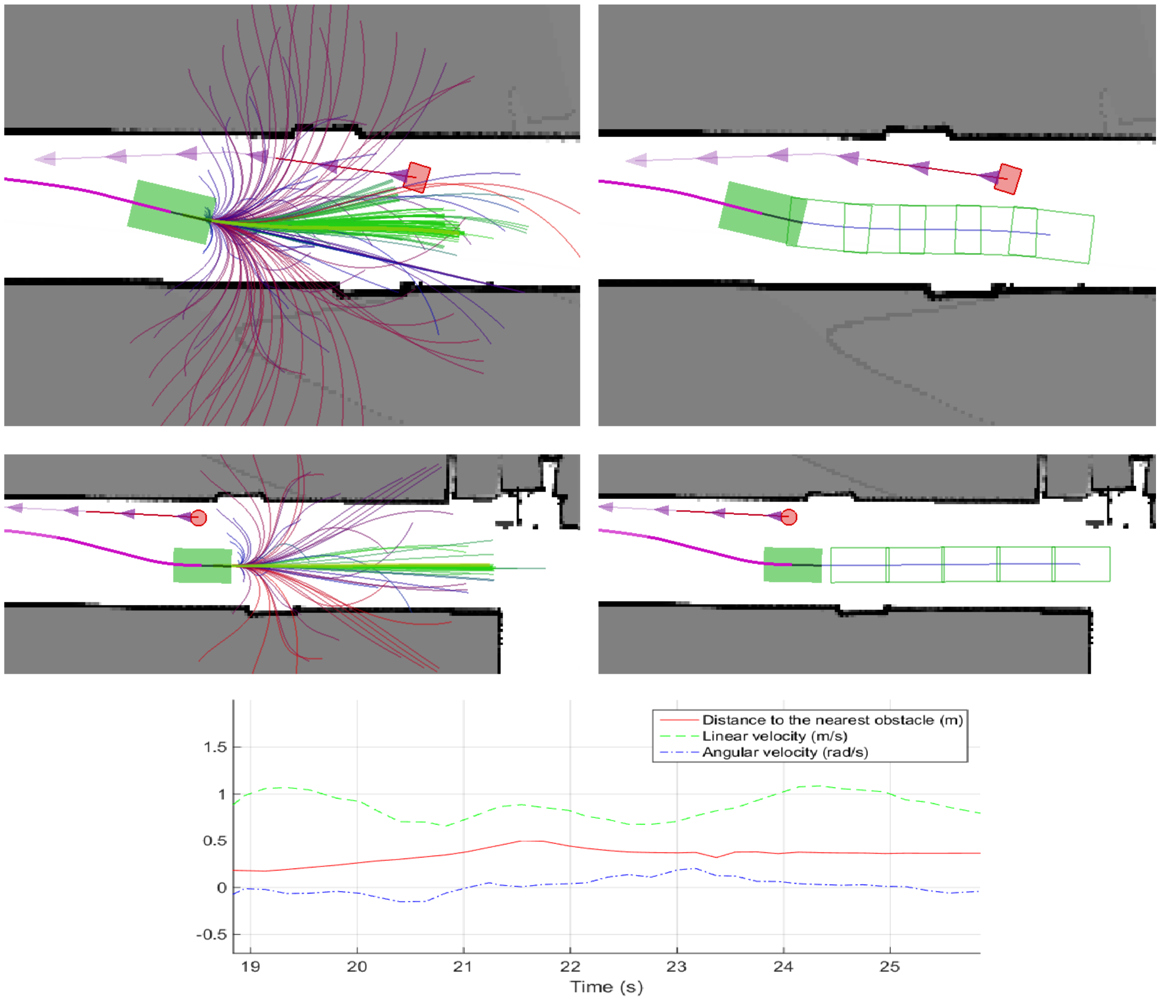
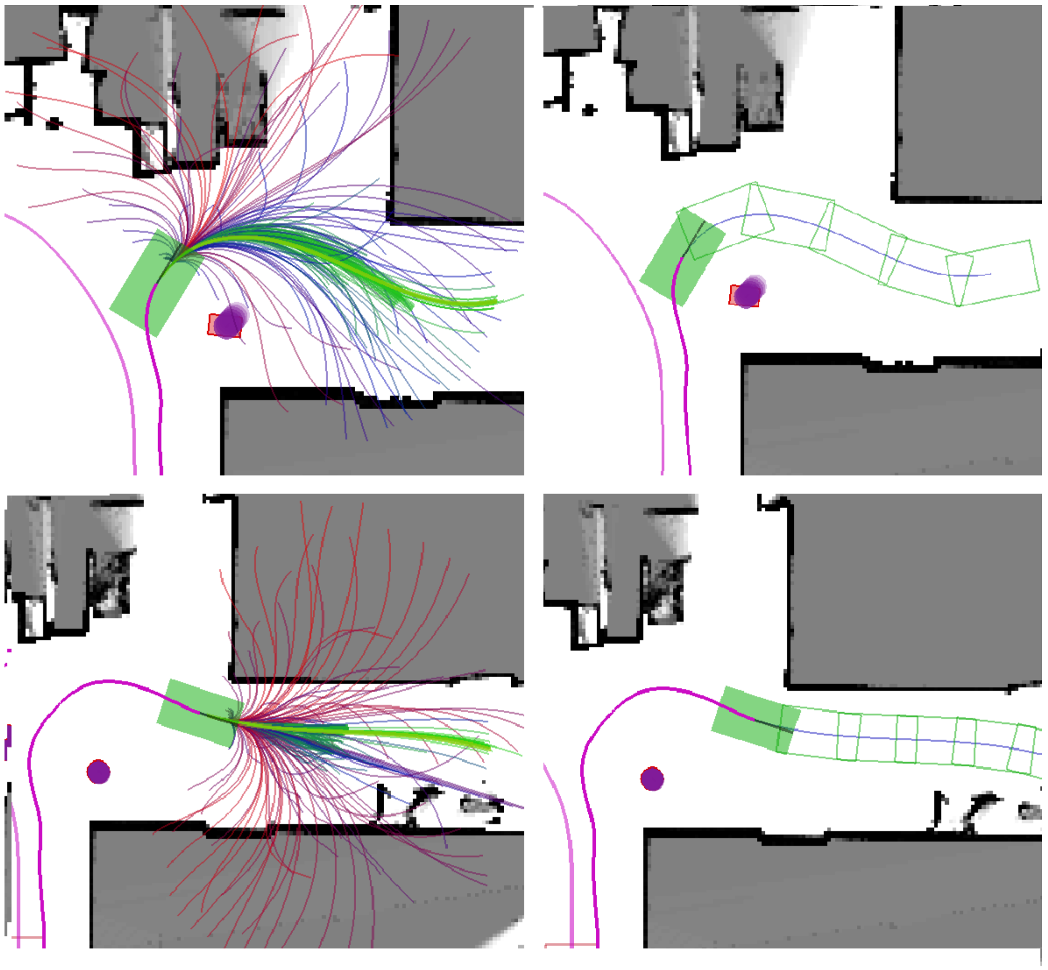


Figure 6.14: Snapshot of the policy and trajectory optimization. The robot avoids an oncoming pedestrian.

Robot safely avoids oncoming pedestrian in a narrow corridor.
Top: 364 trajectories evaluated (green-blue-red color gradient indicates better to worse) in 56ms (left) to find the optimal solution (right) at $t = 22$s. Estimated robot and pedestrian pose, green rectangle and purple triangles are drawn over the estimated trajectories at 1 sec interval.
Middle: 157 trajectories evaluated in 40ms (left) and the planned trajectory (right) after the robot has passed the person at $t = 24$s. Th robot speeds up smoothly.
Bottom: Distance to the nearest obstacle, and linear and angular speed of the robot.



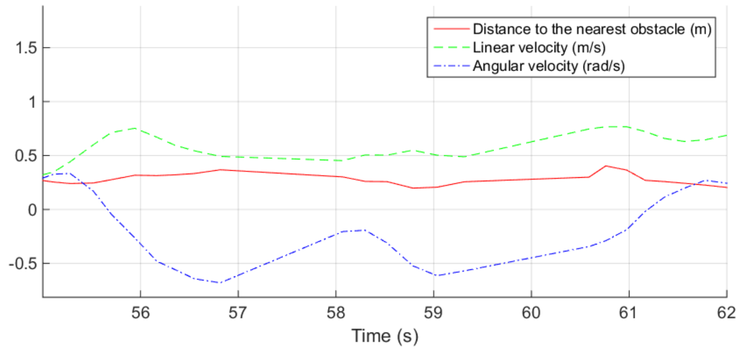


Figure 6.15: Snapshot of the policy and trajectory optimization. The robot moves around a standing pedestrian.

Robot avoids a standing pedestrian while taking corner to the right.
Top: 233 trajectories are evaluated in 40ms (left), and the robot plans to make a large curve, around $v \simeq 0.5$m/s and $\omega \simeq -0.5$rad/s (right), at $t = 59$s
Middle: 392 trajectories evaluated in 60ms (left), and the planned trajectory (right) at $t = 61$ s.
Bottom: Distance to the nearest obstacle, and linear and angular speed of the robot.

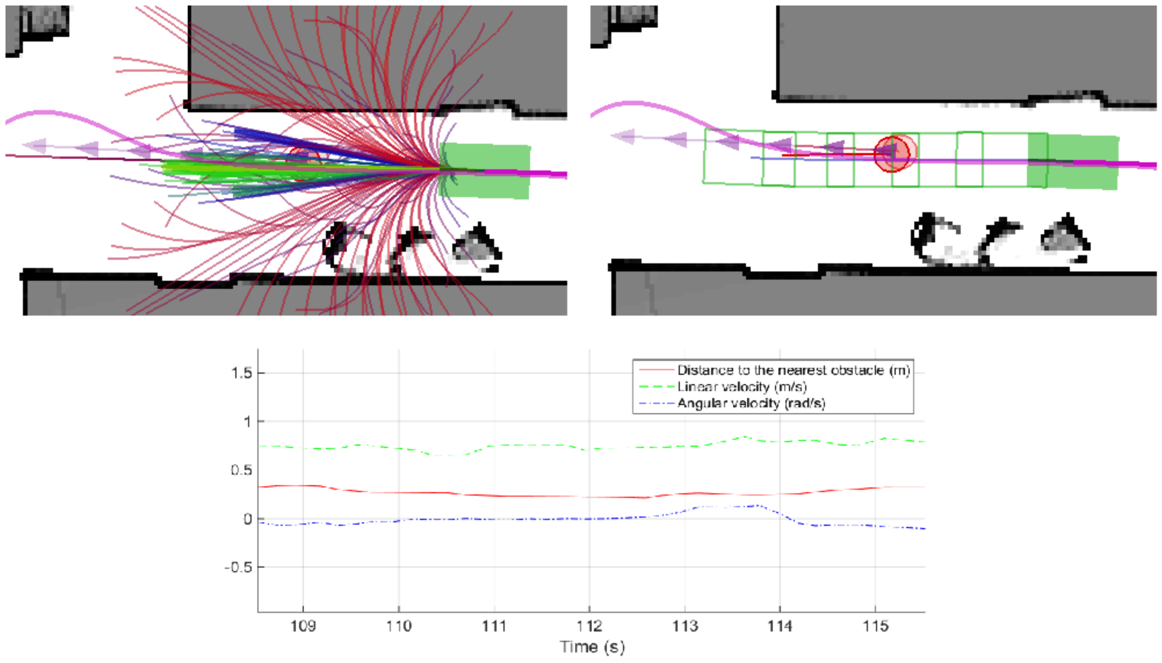


Figure 6.16: Snapshot of the policy and trajectory optimization. The robot moves behind a slow-moving pedestrian.

Top: Robot apparently following the person in front, while moving toward a goal pose to the left side of the map. To make the maximal progress without collision, the robot has to trail the pedestrian in front.
Bottom: Distance to the nearest obstacle, and linear and angular velocity of the robot.

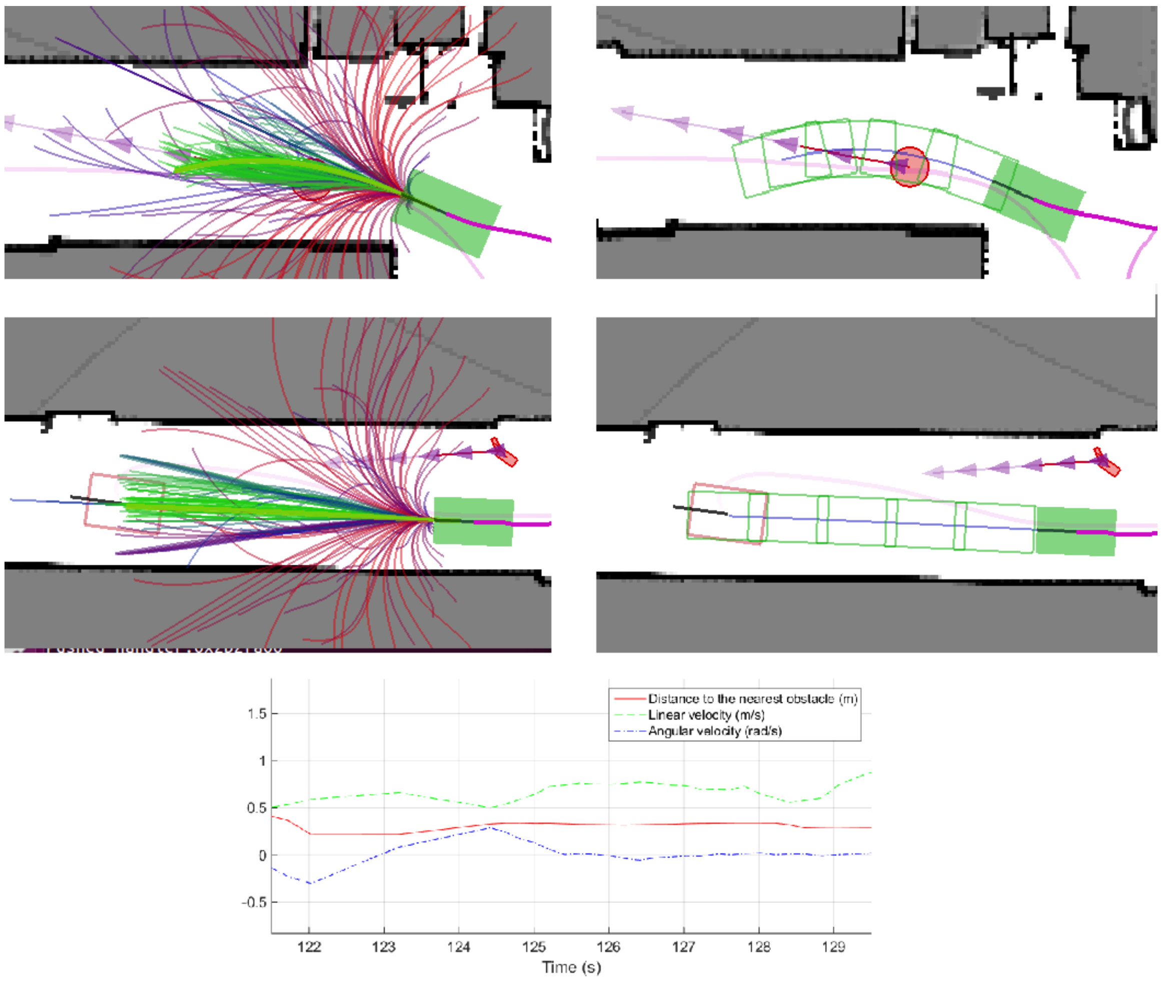


Figure 6.17: Snapshot of the policy and trajectory optimization. The robot overtakes a slow-moving pedestrian.

Top: Robot decides to overtake a pedestrian. 411 candidate trajectories are evaluated in 63 sec, with initial robot speed at $v = 0.68$ m/s.
Middle: Robot safely passes the pedestrian and approaches the target pose at $v = 0.93$ m/s.
Bottom: Distance and velocity plot.

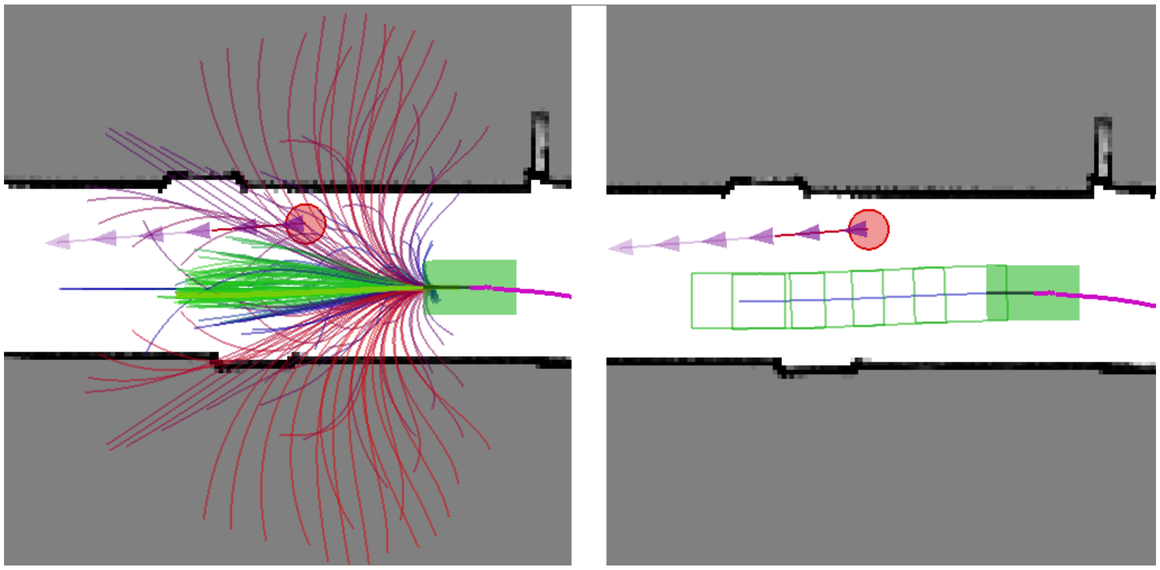


Figure 6.18: Snapshot of the policy and trajectory optimization that results in an apparent following behavior.

Robot apparently moving alongside a pedestrian. The situation is essentially the same as Fig. 6.17, but here the overall weight for the stage cost (quadratic in velocities) is twice ($c_v = 0.8$) the nominal value, rendering the preferred speed to be at $v \simeq 0.6$ m/s. In general, robot behavior appears much less aggressive with higher cost on velocities, and vice versa, as expected.

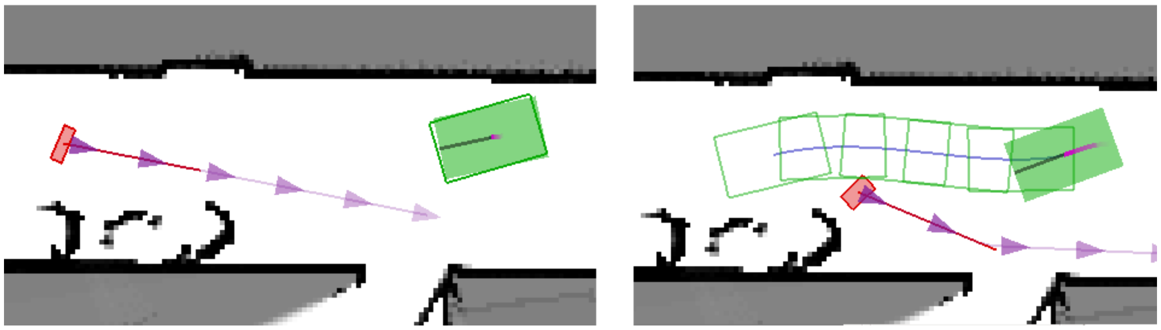


Figure 6.19: Snapshot of the policy and trajectory optimization that allows the robot to wait for a person to pass.

Robot can also wait for a person at short narrow passage as a result of direct optimization. If moving forward will be likely to result in collision (Left), the optimal solution is to stay still. As soon as the person clears however, the robot decides to move (Right).

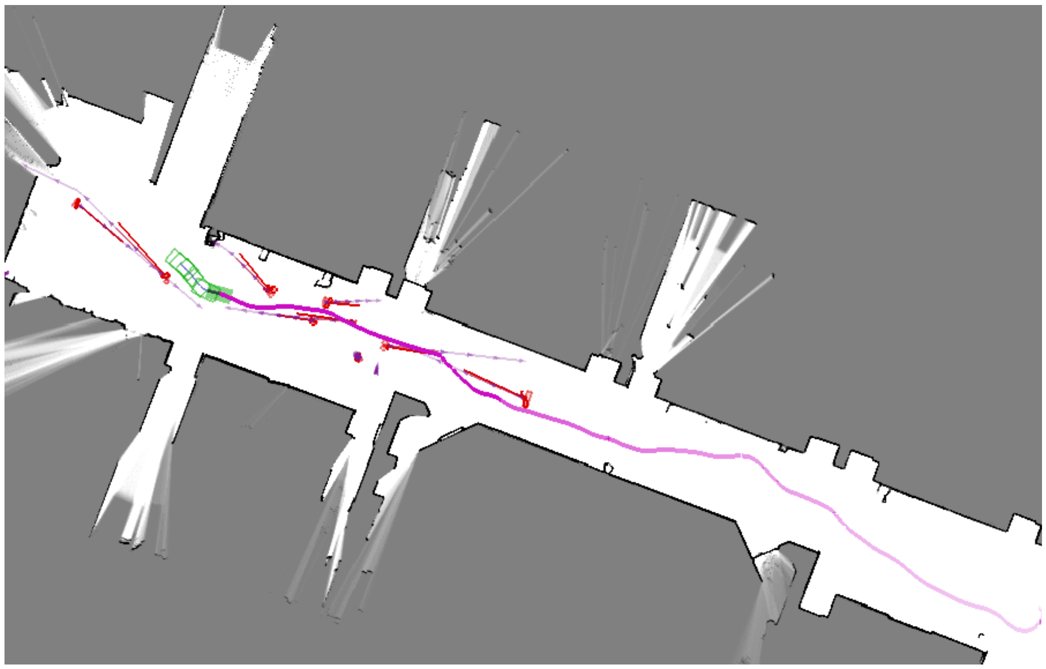


Figure 6.20: Trace of the robot ($\simeq$ 40m) navigating autonomously among modestly dense crowd in a large ($\simeq$ 60x6m) hallway.

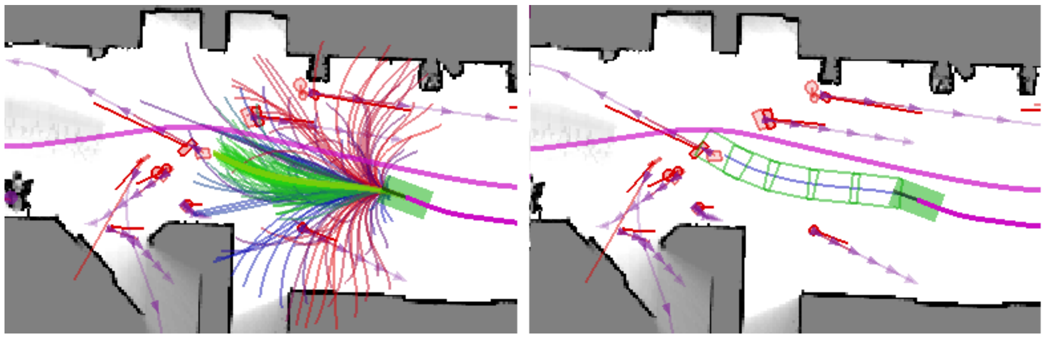


Figure 6.21: The robot finding way through multiple pedestrians.

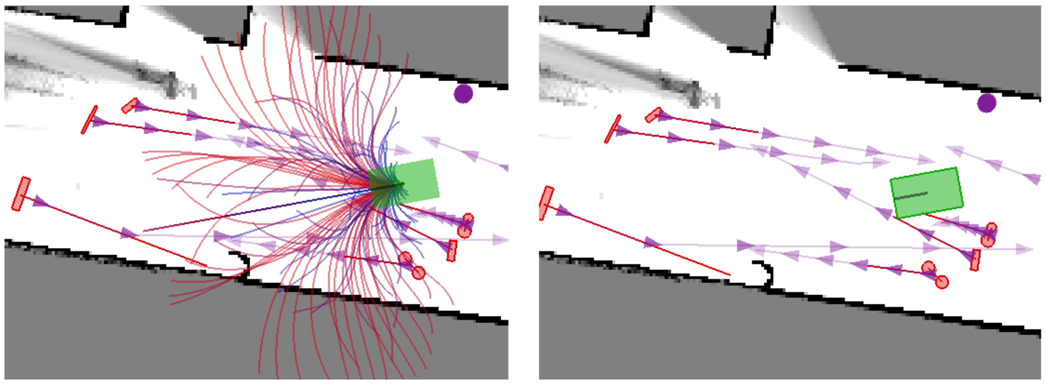


Figure 6.22: The optimal solution is to stay still when it is not safe to move.

CHAPTER VII

Conclusion and Future Work

Graceful navigation in the real world is difficult because of dynamic constraints, modeling uncertainty, noisy measurements, partial sensory data, and real-time computation requirements. In this thesis, we have presented our MPEPC algorithm for mobile robot navigation in dynamic and uncertain environments. Our algorithm is a stochastic model predictive control which determines the optimal control policy at each time step to minimize the expected cost over a specified prediction horizon. The cost function is well defined so that the desired performance can be achieved across a wide range of real-life situations, and the policy is compactly parameterized so that the computational load is minimal and can be run in real-time on a single core of a typical laptop.

This depends on four specific technical contributions. We define our *expected cost* so that we can directly incorporate the time-varying uncertain constraints and the probability of violating those constraints into the cost function, which tends to create a smooth cost surface that is easy to optimize over. The dimensionality reduction of this problem critically depends on the *policy and closed-loop trajectory parameterization* based on our Lyapunov-based feedback control law, which we have developed for graceful motion of differential wheeled mobile robots. The stability of this stochastic model predictive control critically depends on our *non-holonomic distance function*, which is also a Control-Lyapunov function for unicycle-type vehicles. We also develop an *accurate model* of the robot for accurate forward prediction.

We demonstrate that our method generates graceful (safe, smooth, comfortable, fast, and intuitive) and customizable robot behavior in physical environments with pedestrians in real time. The work presented in this thesis extends the state-of-the-art in analytic control of mobile robots, sampling-based optimal path planning, and stochastic model predictive control. The proposed algorithm addresses the difficult problem of navigating in uncertain and dynamic environment safely and comfortably while avoiding hazards, which is a necessary task for autonomous passenger vehicles. We believe that our work is a significant step toward safe and reliable autonomous navigation that is acceptable to human users.

There are several important future research directions that could improve the performance of the algorithm. First, since the performance of model predictive control in general critically depends on the quality of the model, it would be ideal if we could learn an accurate model of the robot over time. This will also greatly facilitate the implementation of the algorithm to different physical platforms. Second, the algorithm can also benefit greatly from more accurate predictions of other agents' future motion. This will involve intention estimation and agent-to-agent interaction modeling. Third, we note that the proper quantification of user preference and comfort, and a quantifiable benchmark for the quality of motion in general are other open areas. These will require further user studies.

APPENDICES

APPENDIX A

System Identification, State Estimation, and Low-Level Control of a Physical Wheelchair

[1] Here, we describe the system identification, state estimation and simulation, and model-based feedforward-feedback velocity controller for our physical robot. The processes described in this appendix provide necessary components for implementation of our proposed motion planning and control algorithm to a physical system.

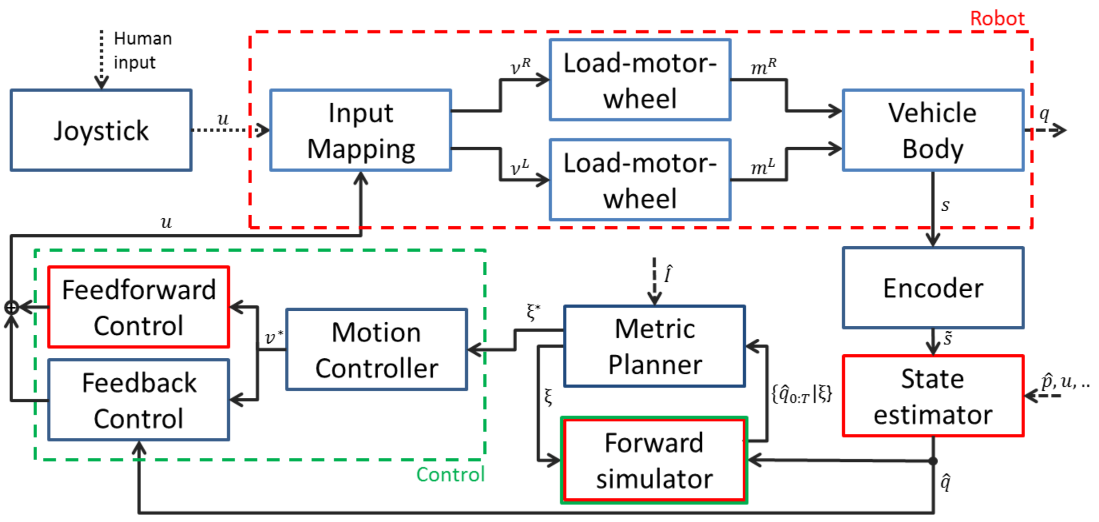


Figure A.1: Vulcan system diagram.

(Best viewed in color) Boxes (processes) in solid red lines uses the robot model. u is the joystick position, ν^R and ν^L are the right and the left motor input, m^R and m^L are the right and the left wheel/motor states, $\tilde{s}$ is the measured displacements of the wheels, $\hat{p}$ is the estimated robot position, q is the full robot state, $\{q\}$ is the state trajectory, $\hat{I}$ is relevant information of the environment, ζ is parameterization of a (pose-stabilizing) control law, and v^* is a target velocity.

[1]This is a revised (and extended) presentation of *Park* (2014).

The physical body of our robot is a commercially available powered wheelchair retrofitted with lidar sensors, an internal measurement unit, and odometers. Modelling the dynamics of this system was not trivial, partly because motor signals and wheel accelerations were not observable (although detectable) due to lack of sensors, and partly due to the usual difficulty of modelling and incorporating the non-linear friction in the system. We note that to achieve desired accuracy in the model it was necessary to incorporate friction and motor saturation in the system. An overview our implementation is illustrated in Fig. A.1.

The model and the system identification process are described in Section A.1, and we show that our physics based model can accurately represent the dynamics of the real system. This in turn allows the high-fidelity robot state estimation and model predictive simulation in Section A.2, and the model-based feedforward-feedback control of wheel velocities in Section A.3.

A.1 Modelling Vehicle Dynamics with Friction

Here, we describe our model of joystick-controlled differential-drive electric powered wheelchair, Vulcan (Fig. 3.7). This driving platform, Quantum6000z from Pride Mobility, is proprietary and the specific information about internal parameters are not available. Our discrete-time model of the vehicle (Fig. A.2) is composed of three subsystems: (i) a mapping from joystick positions into motor commands; (ii) load-motor-wheel dynamics model; and (iii) vehicle body which maps wheel speeds into vehicle velocities and integrates poses.

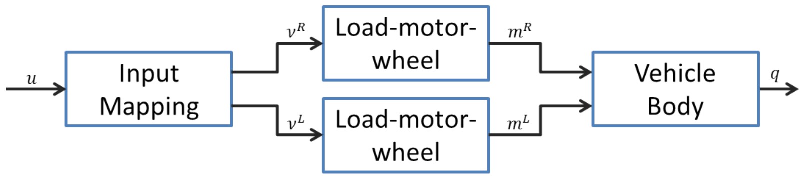


Figure A.2: System diagram for our vehicle model.

u is the external control input (joystick), ν is the motor input, m is the motor/wheel state, and q is the full robot state. Superscripts R/L denote right/left.

The full state vector for the robot consists of the pose $p = [x\, y\, \theta]^T$ and the two motor/wheel states $m = [\dot{s}, \ddot{s}]$ where s is displacement of a wheel, i.e., $q \equiv [p\, m^R\, m^L]^T = [x\, y\, \phi\, \dot{s}^R\, \ddot{s}^R\, \dot{s}^L\, \ddot{s}^L]^T$, where the superscripts R and L denote the right and the left wheel of the differential drive vehicle. Note that there exists a bijective relation between the vehicle velocities $[v\, \omega]$ and the wheel velocities $[\dot{s}^R\, \dot{s}^L]$ (see (A.7)), so $[x\, y\, \phi\, \dot{s}^R\, \ddot{s}^R\, \dot{s}^L\, \ddot{s}^L] \sim$

$[x\, y\, \phi\, v\, \omega\, \ddot{s}^R\, \ddot{s}^L]$.

We will show that the dynamics of the system can be closely approximated by a difference equation (A.1), that is parametrized by 8 positive constant, c_0, c_1, c_2, α, β, γ, μ, and L. The meaning of the constants will be discussed later. Formally, we write

$$q_{k+1} = f(q_k, u_k) = f(q_k, u_k;\ c_0, c_1, c_2, \alpha, \beta, \gamma, \mu, L) \tag{A.1}$$

where q_k and the u_k are the state and control input of the robot at the k-th step at time t_k, and q_{k+1} is the state at the next time step. The control input $u_k = [u_k^f\, u_k^l]^T$ consists of the forward (u^f) and the lateral (u^l) joystick positions, which are the only available control input to the system. The robot is equipped with encoders and SLAM, which gives us observations on individual wheel speeds and pose.

A.1.1 Input Mapping

Empirically, we have found the following non-linear model for joystick-input-to-motor-input mapping work well with data when combined with the other subsystems:

$$\begin{bmatrix} \nu^R \\ \nu^L \end{bmatrix} = \begin{bmatrix} 1 & g_u(u^f) \\ 1 & -g_u(u^f) \end{bmatrix} \begin{bmatrix} u^f \\ u^l \end{bmatrix} \tag{A.2}$$

where ν^R and ν^L are unitless inputs to the right and the left motors, u^f and u^l are the forward and lateral joystick positions, and

$$g_u(u^f) = \begin{cases} c_0 & \text{if } u^f = 0 \\ c_1(1 - c_2|u^f|) & \text{otherwise} \end{cases} \tag{A.3}$$

is a non-linear scaling factor for the lateral joystick position which models interaction between forward-backward and left-right joystick commands, where c_0, c_1 and c_2 are non-zero positive constants that parameterize this mapping.

A.1.2 Load-motor-wheel Subsystem

This subsection describes a model for the load-motor-wheel subsystem, i.e., an electric motor connected to a wheel under a constant unknown load. Let s_k be the displacement of the wheel, and ν_k is the unitless input to the motor which is converted to jerk (which is proportional to Voltage for electric motors) via some gain. Then we can write the following model for the load-motor-wheel subsystem,

$$\begin{bmatrix} \dot{s}_{k+1} \\ \ddot{s}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h_k \\ -\beta h_k & 1-\gamma h_k \end{bmatrix} \begin{bmatrix} \dot{s}_k \\ \ddot{s}_k \end{bmatrix} + \begin{bmatrix} h_k \\ 0 \end{bmatrix} g_r(\dot{s}_k, \ddot{s}_k; \mu) + \begin{bmatrix} 0 \\ \alpha h_k \end{bmatrix} \nu_k \quad \text{(A.4)}$$

where $\dot{s}_k$ and $\ddot{s}_k$ are speed and acceleration of the wheel which constitute the state vector $m_k \equiv [\dot{s}_k \ddot{s}_k]^T$ for this subsystem, which we call wheel/motor state, and $h_k \equiv t_{k+1} - t_k$ is the time interval between k- and $k+1$-th time step. The positive constants α, β and γ models the input gain, velocity- and current-induced resistance for standard DC-motors, respectively, and $g_r(\cdot)$ models acceleration from the load-induced mechanical friction *Kikuuwe et al.* (2005)[2]

$$g_r(\dot{s}_k, \ddot{s}_k; \mu) = \begin{cases} \bar{\mu}(\cdot) & \text{if } |\bar{\mu}(\cdot)| \leq \mu \\ \text{sgn}(\bar{\mu}(\cdot))\,\mu & \text{otherwise} \end{cases} \quad \text{(A.5)}$$

where

$$\bar{\mu}(\dot{s}_k, \ddot{s}_k) \equiv -\frac{\dot{s}_k}{h_k} - \ddot{s}_k \quad \text{(A.6)}$$

and the positive constant μ is a parameter for the maximum friction-induced acceleration which is a function of unknown but constant load and friction coefficient.

In this model, there are four parameters to fit for each wheel-motor-load subsystem, α, β, γ, and μ, which represents input gain, and velocity-, current-, and load- induced resistances. We assume identical wheel, motor and load for the right and the left subsystems.

A.1.3 Differential-drive Motion Model

For the motion of differential-drive vehicles on 2D plane, we can write

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix} \begin{bmatrix} \dot{s}^R \\ \dot{s}^L \end{bmatrix} \quad \text{(A.7)}$$

where v and ω are linear and angular velocity of the rigid body, L is the length of the axle between two denote the length of the axle between the two drive wheels, and $\dot{s}^R$ and $\dot{s}^L$ are the speed of the right and the left wheel.

Assuming no-slip condition, the linear and angular velocities are inputs to the following

[2]This paper *Kikuuwe et al.* (2005) is an excellent reference for different friction models for discrete-time systems. The key feature of (A.5)-(A.6) is the speed drops exactly to zero in a single step if $|\bar{\mu}(\cdot)| \leq \mu$, removing the need of special event detection and special rules near zero speed, which can often become unnecessarily complicated. Here we start with a simple constant Coulomb friction model, but it can be extended to include arbitrary velocity dependencies by modifying (A.6) if needed.

motion model of the vehicle:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \phi_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \bar{v}_k h_k \cos(\phi_k + \frac{1}{2}\bar{\omega}_k h_k) \\ y_k + \bar{v}_k h_k \sin(\phi_k + \frac{1}{2}\bar{\omega}_k h_k) \\ \phi_k + \bar{\omega}_k h_k \end{bmatrix} \tag{A.8}$$

where $p_k \equiv [x_k \, y_k \, \phi_k]^T$ is the pose of the vehicle at time t_k, and $\bar{v}_k$ and $\bar{\omega}_k$ are average linear and angular velocity over the time interval h_k, approximated with $\bar{v} = \frac{1}{2}(v_k + v_{k+1})$ and $\bar{\omega} = \frac{1}{2}(\omega_k + \omega_{k+1})$. This half turn-drive-half turn approach closely approximates real vehicle motion (*Wang*, 1988).

A.1.4 Evaluation

Fig. A.3-A.5 shows comparison between linear and angular velocities measured from the encoders (blue) to the velocities predicted by the model (magenta). The model is only given the initial state $q = [0, 0, 0, 0, 0, 0, 0]^T$ and the measured control inputs $t = [0\,945]$ (seconds), and the system model (A.1) is recursively applied to the state and the control input to generate model-based prediction of linear and angular velocity. Overall, the prediction stays highly accurate for the entire duration of the dataset, without using any state measurement except the initial condition.

A.2 Simulation and State Estimation

Robot trajectory simulation is a repeated application of (A.1) to a robot state q given some control u, propagating the state through time to obtain an estimate of future trajectory. This propagation is carried out in two steps: First, the motor/wheel states $\hat{m}_k^R$ and $\hat{m}_k^L$ are propagated under the control u_k via (A.2)-(A.6); Then, the pose $\hat{p}_k$ is propagated under average velocities $\bar{v}_k$ and $\bar{\omega}_k$ via (A.8), where the average velocities are computed from velocities mapped from the previous and new motor/wheel state estimates via (A.7). Note that all mappings are well defined and the uncertainties can also be propagated.

For state estimation, we use Kalman filters. We employ three separate filters, one each for prediction and correction of the states m^R, m^L and p. Relevant observations are encoder readings $\tilde{s}^R$ and $\tilde{s}^L$ from each wheel, and the pose estimate $\hat{p}$ from the localization module (Fig. A.6).

The equations (A.2)-(A.8) are the process models for the prediction step of the Kalman filters. For m^R and m^L, the process model consists of (A.2)-(A.6), and the observation

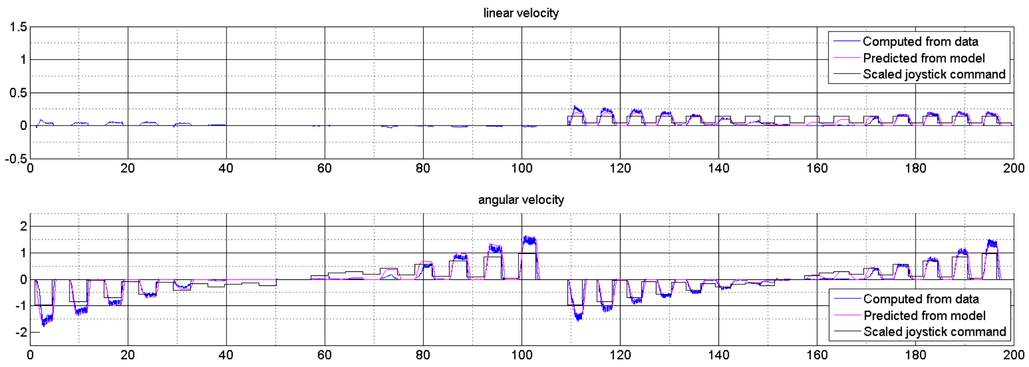


Figure A.3: The model prediction vs. measurements at low velocities with step inputs.

(Best viewed in color) The model prediction (magenta) vs. measurement (blue) at low velocities with step inputs. The model prediction is using state measurement only at $t = 0$, and recursively simulated forward via the proposed model.

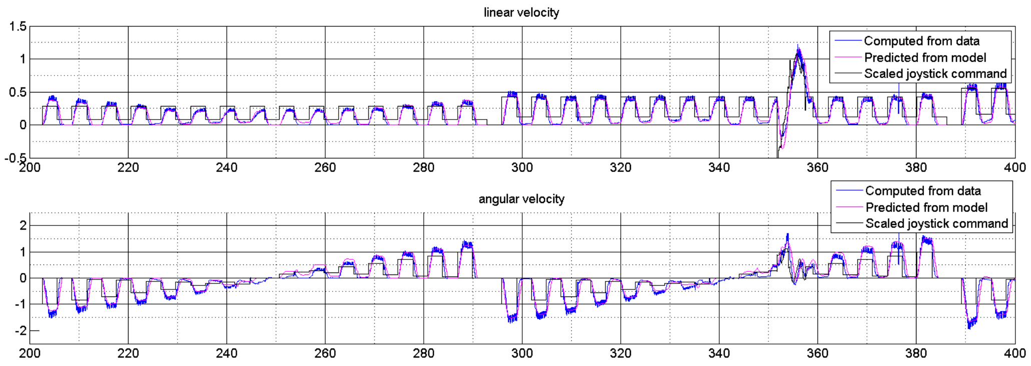


Figure A.4: The model prediction vs. measurements at higher velocities with step inputs.

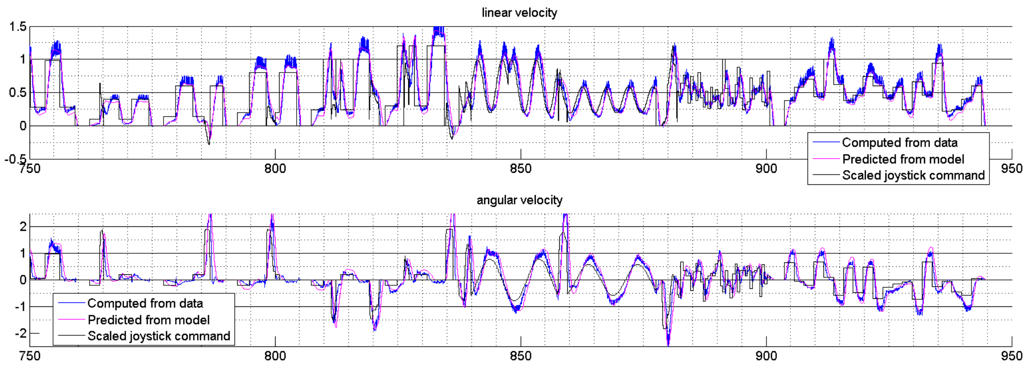


Figure A.5: The model prediction vs. measurements with random step and sinusoidal inputs.

The model is able to predict the robot velocities accurately for the entire duration of the dataset (945 seconds), without using any state measurement except the initial condition.

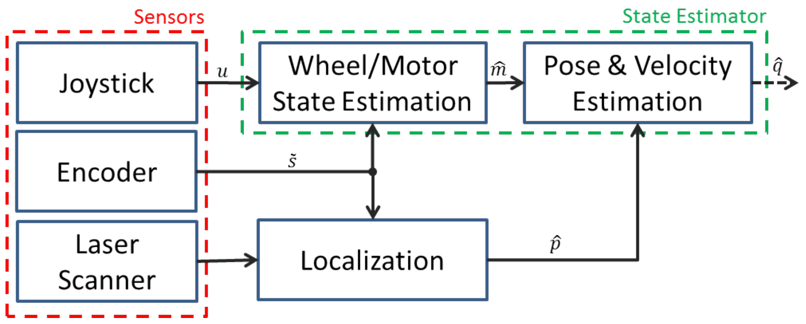


Figure A.6: State estimator diagram.

The joystick positions u propagates the wheel/motor states which is corrected with encoder readings $\tilde{s}$. The estimated wheel/motor states $\hat{m}$ is mapped into vehicle velocities, treated as input to the vehicle kinematic model and used to update vehicle pose using the estimate of the robot pose $\hat{p}$ from localization module.

model[3] using the encoder readings $\tilde{s}$ is

$$\hat{\dot{s}}_k = \frac{\tilde{s}_{k+1} - \tilde{s}_k}{h_k} \tag{A.9}$$

For p, the process model consists of (A.7)-(A.8), and the observation model is identity, as the full pose estimate and the covariance is available as the result of the localization.

Note that, in all our equations, we explicitly state that the time interval h_k can be time-varying. That is because in practice time interval between observations from sensors is almost never constant, and have to be re-evaluated at each update using the sensor time stamps.

A.3 Model-based Velocity Control

Here we describe a method to stabilize the velocity of the vehicle to a desired value. For our system, we know from data that constant input (joystick position) corresponds to certain steady-state velocity, which is typical for a commercially developed platform intended for human use. To use that for our benefit, we begin with the steady-state analysis of the load-motor-wheel subsystem (A.4), and construct a feedforward-feedback control design.

[3]For the load-motor-wheel subsystem (A.4), the acceleration $\ddot{s}$ is not directly observable but it is detectable via observation of the speed $\dot{s}$, thus it is possible to estimate the full wheel/motor state.

A.3.1 Steady-state Analysis

Suppose the load-motor-wheel subsystem (A.4) is in steady state so that $\dot{s}_k = \dot{s}_\infty$ and $\ddot{s}_k = \ddot{s}_\infty$ for $^\forall k$ under some constant input ν. To begin, assume the vehicle is moving forward at constant speed, i.e. $\dot{s}, \ddot{s} > 0$ and $|\bar{\mu}(\cdot)| > \mu$, then we have from (A.4)-(A.5)

$$\begin{bmatrix} \dot{s}_\infty \\ \ddot{s}_\infty \end{bmatrix} = \begin{bmatrix} 1 & h \\ -\beta h & 1-\gamma h \end{bmatrix} \begin{bmatrix} \dot{s}_\infty \\ \ddot{s}_\infty \end{bmatrix} + \begin{bmatrix} -\mu h \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha h \end{bmatrix} \nu_\infty \tag{A.10}$$

so that

$$\begin{cases} \dot{s}_\infty &= \frac{\alpha}{\beta}\nu_\infty - \frac{\mu\gamma}{\beta} \quad (\nu_\infty > \frac{\mu\gamma}{\alpha}) \\ \ddot{s}_\infty &= \mu \end{cases} \tag{A.11}$$

which means the steady-state speed is a function of constant input, and when in steady state the constant motor acceleration is cancelled out by constant friction, as expected.

Performing this analysis for all cases, we can write

$$\dot{s}_\infty = \begin{cases} \frac{\alpha}{\beta}\nu_\infty - \frac{\mu\gamma}{\beta} & \text{if } \nu_\infty > \frac{\mu\gamma}{\alpha} \\ \frac{\alpha}{\beta}\nu_\infty + \frac{\mu\gamma}{\beta} & \text{if } \nu_\infty < -\frac{\mu\gamma}{\alpha} \\ 0 & \text{otherwise} \end{cases} \tag{A.12}$$

and

$$\nu_\infty \begin{cases} = \frac{\beta}{\alpha}\dot{s}_\infty + \frac{\mu\gamma}{\alpha} & \text{if } \dot{s}_\infty > 0 \\ \in [-\frac{\mu\gamma}{\alpha}, \frac{\mu\gamma}{\alpha}] & \text{if } \dot{s}_\infty = 0 \\ = \frac{\beta}{\alpha}\dot{s}_\infty - \frac{\mu\gamma}{\alpha} & \text{if } \dot{s}_\infty < 0. \end{cases} \tag{A.13}$$

Eq. (A.12), and Eq. (A.13) with collapsed nullspace ($\nu_\infty = 0$ if $\dot{s}_\infty = 0$) are shown in Fig. A.7.

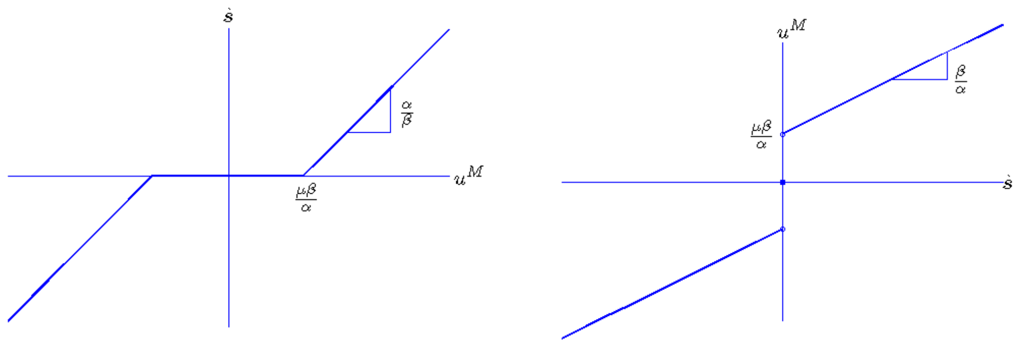


Figure A.7: Steady-state wheel speeds and steady-state motor inputs.
Left: Steady state motor input vs. speed. **Right:** Steady state speed vs. motor input.

A.3.2 Feedforward Control

Eq. (A.13) allows us to construct a model-based feedforward control. Suppose we want to achieve a steady-state velocity v^*, ω^*. By inverting (A.7), we get the corresponding steady-state wheel speeds

$$\begin{bmatrix} \dot{s}^{*R} \\ \dot{s}^{*L} \end{bmatrix} = \begin{bmatrix} 1 & \frac{L}{2} \\ 1 & -\frac{L}{2} \end{bmatrix} \begin{bmatrix} v^* \\ \omega^* \end{bmatrix} \tag{A.14}$$

which then can be mapped to the corresponding command ν^* for each motor, with

$$\nu^* = \begin{cases} \frac{\beta}{\alpha}\,\dot{s}^* + \operatorname{sgn}(\dot{s}^*)\frac{\mu\gamma}{\alpha} & \text{if } \dot{s}^* \neq 0 \\ 0 & \dot{s}^* = 0 \end{cases} \tag{A.15}$$

which is constructed from (A.13) by substituting $\dot{s}_\infty$ with $\dot{s}^*$ and taking $\nu = 0$ if $\dot{s}^* = 0$. Now, inverting the joystick-to-motor mapping (A.2), we get

$$u^f_{\text{feedforward}} = \frac{1}{2}(\nu^{*R} + \nu^{*L}) \tag{A.16}$$

$$u^l_{\text{feedforward}} = \frac{1}{2g_u(u_f)}(\nu^{*R} - \nu^{*L}) \tag{A.17}$$

where ν^{*R} and ν^{*L} are the commands for the right and the left motors obtained by (A.15). Eq.(A.14)-(A.17) maps the steady-state velocities to corresponding joystick commands, assuming accurate model.

A.3.3 Feedforward-feedback Control

We rely on the feedforward control to achieve the desired steady state, but it is often desirable to have additional feedback control to improve transient response of the system or to reduce error due to imperfect model. We have implemented a simple saturation-imposed P-controller for that purpose. Combined with the feedforward control (A.16)-(A.17), the full controller can be written as

$$u^f = \frac{1}{2}(\nu^{*R} + \nu^{*L}) + P_v\, g_v(v^* - \hat{v}) \tag{A.18}$$

$$u^l = \frac{1}{2g_u(u_f)}(\nu^{*R} - \nu^{*L}) + P_\omega\, g_\omega(\omega^* - \hat{\omega}) \tag{A.19}$$

where the P_v and P_ω are the gains, the $\hat{v}$ and $\hat{\omega}$ are current estimate of the linear and angular velocities, and

$$\begin{aligned} g_v(v^* - \hat{v}) &= \text{sgn}(v^* - \hat{v}) \cdot \min(|v^* - \hat{v}|,\, u^f_{\max}/P_v) \\ g_\omega(\omega^* - \hat{\omega}) &= \text{sgn}(\omega^* - \hat{\omega}) \cdot \min(|\omega^* - \hat{\omega}|,\, u^l_{\max}/P_\omega) \end{aligned}$$

where $u^f_{\max}$ and $u^l_{\max}$ are user-imposed saturation bounds.

BIBLIOGRAPHY

BIBLIOGRAPHY

Aguiar, A., and A. Pascoal (2007), Dynamic positioning and way-point tracking of underactuated AUVs in the presence of ocean currents, *International Journal of Control*, *80*(7), 1092–1108.

Aicardi, M., G. Casalino, A. Bicchi, and A. Balestrino (1995), Closed loop steering of unicycle-like vehicles via Lyapunov techniques, *IEEE Robotics and Automation Magazine*, *2*(1), 27–35.

Aoude, G., B. Luders, J. Joseph, N. Roy, and J. How (2013), Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns, *Autonomous Robots*, *35*(1), 51–76.

Bacha, A., et al. (2008), Odin: Team Victor Tango's entry in the DARPA urban challenge, *Journal of Field Robotics*, *25*(8), 467–492.

Bakaric, V., Z. Vukic, and R. Antonic (2004), Improved basic planar algorithm of vehicle guidance through waypoints by the line of sight, in *First Int. Symposium on Control, Communications and Signal Processing*, pp. 541–544.

Bentley, J. L. (1975), Multidimensional binary search trees used for associative searching, *Communications of ACM*, *18*(9), 509–517.

Bertrand, S., T. Hamel, and H. Piet-Lahanier (2006), Performance improvement of an adaptive controller using model predictive control : Application to an UAV model, in *4th IFAC Symposium on Mechatronic Systems*, vol. 4, pp. 770–775.

Blackmore, L., H. Li, and B. Williams (2006), A probabilistic approach to optimal robust path planning with obstacles, in *American Control Conference*, pp. 7–13.

Bohren, J., et al. (2008), Little ben: the ben franklin racing team's entry in the 2007 darpa urban challenge, *Journal of Field Robotics*, *25*(9), 598–614.

Borenstein, J., and Y. Koren (1991), The vector field histogram-fast obstacle avoidance for mobile robots, *IEEE Transactions on Robotics and Automation*, *7*(3), 278–288.

Breivik, M., and T. Fossen (2008), Guidance laws for planar motion control, in *47th IEEE Conference on Decision and Control*, pp. 570–577.

Brockett, R. W. (1983), Asymptotic stability and feedback stabilization, in *Differential Geometric Control Theory*, edited by R. W. Brockett, R. S. Millman, and H. J. Sussmann, pp. 181–191, Birkhauser, Boston, MA.

Burridge, R., A. Rizzi, and D. Koditschek (1999), Sequential composition of dynamically dexterous robot behaviors, *International Journal of Robotics Resaech*, *18*(6), 534–555.

Charnes, A., and W. W. Cooper (1959), Chance-constrained programming, *Management Science*, *6*(1), 73–79.

Cheng, P., and S. M. LaValle (2001), Reducing metric sensitivity in randomized trajectory design, in *2001 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 43–48.

Choset, H., K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun (2005), *Principles of Robot Motion: Theory, Algorithms, and Implementations*, The MIT Press, Cambridge, MA.

Chou, C. T., J.-Y. Li, M.-F. Chang, and L. C. Fu (2011), Multi-robot cooperation based human tracking system using laser range finder, in *2011 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 532–537.

Dijkstra, E. (1959), A note on two problems in connexion with graphs, *Numerische Mathematik*, *1*(1), 269–271.

Dolgov, D., S. Thrun, M. Montemerlo, and J. Diebel (2010), Path planning for autonomous vehicles in unknown semi-structured environments, *The International Journal of Robotics Research*, *29*(5), 485–501.

Droge, G., and M. Egerstedt (2011), Adaptive look-ahead for robotic navigation in unknown environments, in *2011 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 1134–1139.

Du Toit, N., and J. Burdick (2011), Probabilistic collision checking with chance constratins, *IEEE Transactions on Robotics*, *27*(4), 809–815.

Du Toit, N., and J. Burdick (2012), Robot motion planning in dynamic, uncertain environments, *IEEE Transactions on Robotics*, *28*(1), 101–115.

Feldman, A. G., and M. F. Levin (2009), The equilibrium-point hypothesis past, present and future, in *Progress in Motor Control*, *Advances in Experimental Medicine and Biology*, vol. 629, edited by D. Sternad, pp. 699–726, Springer US.

Ferguson, D., and A. Stentz (2007), Field D*: An interpolation-based path planner and replanner, in *Robotics Research*, *Springer Tracts in Advanced Robotics*, vol. 28, edited by S. Thrun, R. Brooks, and H. Durrant-Whyte, pp. 239–253, Springer Berlin Heidelberg.

Ferguson, D., T. M. Howard, and M. Likhachev (2008), Motion planning in urban environments, *Journal of Field Robotics*, *25*(11-12), 939–960.

Fiorini, P., and Z. Shiller (1998), Motion planning in dynamic environments using velocity obstacles, *The International Journal of Robotics Research*, *17*(7), 760–772.

Fox, D., W. Burgard, and S. Thrun (1997), The dynamic window approach to collision avoidance, *IEEE Robotics Automation Magazine*, *4*(1), 23 –33.

Frew, E. (2005), Receding horizon control using random search for UAV navigation with passive, non-cooperative sensing, in *2005 AIAA Guidance, Navigation, and Control Conf. and Exhibit*, pp. 1–13.

Frintrop, S., A. Knigs, F. Hoeller, and D. Schulz (2010), A component-based approach to visual person tracking from a mobile platform, *Intternaltional Journal of Social Robotics*, *2*, 53–62.

Germa, T., F. Lerasle, N. Ouadah, V. Cadenat, and M. Devy (2009), Vision and rfid-based person tracking in crowds from a mobile robot, in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 5591–5596.

Glassman, E., and R. Tedrake (2010), A quadratic regulator-based heuristic for rapidly exploring state space, in *2010 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 5021–5028.

Gockley, R., and M. Mataric (2006), Encouraging physical therapy compliance with a hands-off mobile robot, in *Proceedings of Human-Robot Interaction, Salt Lake City, Utah*, pp. 150–155.

Gockley, R., J. Forlizzi, and R. Simmons (2007), Natural person-following behavior for social robots, in *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pp. 17–24.

Goretkin, G., A. Perez, R. Platt, and G. Konidaris (2013), Optimal sampling-based planning for linear-quadratic kinodynamic systems, in *2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2429–2436.

Grancharova, A., and T. A. Johansen (2012), Nonlinear model predictive control, in *Explicit Nonlinear Model Predictive Control*, *Lecture Notes in Control and Information Sciences*, vol. 429, pp. 39–69, Springer Berlin / Heidelberg.

Green, C., and A. Kelly (2007), Toward optimal sampling in the space of paths, in *13th International Symposium of Robotics Research*, pp. 281–292.

Gulati, S. (2011), A framework for characterization and planning of safe, comfortable, and customizable motion of assistive mobile robots, Ph.D. thesis, The University of Texas at Austin.

Gulati, S., and B. Kuipers (2008), High performance control for graceful motion of an intelligent wheelchair, in *2008 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 3932–3938.

Gulati, S., C. Jhurani, B. Kuipers, and R. Longoria (2009), A framework for planning vomfortable and vustomizable motion of an assistive mobile robot, in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4253–4260.

Hart, P., N. Nilsson, and B. Raphael (1968), A formal basis for the heuristic determination of minimum cost paths, *IEEE Tranactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hemachandra, S., T. Kollar, N. Roy, and S. Teller (2011), Following and interpreting narrated guided tours, in *2011 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2574–2579.

Hoeller, F., D. Schulz, M. Moors, and F. Schneider (2007), Accompanying persons with a mobile robot using motion prediction and probabilistic roadmaps, in *2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 1260–1265.

Horiuchi, T., S. Thompson, S. Kagami, and Y. Ehara (2007), Pedestrian tracking from a mobile robot using a laser range finder, in *IEEE Int. Conf. on Systems, Man and Cybernetics*, pp. 931–936.

Howard, T., C. Green, and A. Kelly (2009), Receding horizon model-predictive control for mobile robot navigation of intricate paths, in *Proceedings of the 7th International Conferences on Field and Service Robotics*, pp. 69–78.

Hu, C.-H., X.-D. Ma, and X.-Z. Dai (2009), Reliable person following approach for mobile robot in indoor environment, in *2009 Int. Conf. on Machine Learning and Cybernetics*, pp. 1815–1821.

Indiveri, G., A. Nücheter, and K. Lingmann (2007), High speed differential drive mobile robot path following control with bounded wheel speed commands, in *2007 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2202–2207.

Jadbabaie, A., J. Yu, and J. Hauser (1999), Stabilizing receding horizon control of nonlinear systems: a control lyapunov function approach, in *American Control Conference*, vol. 3, pp. 1535–1539.

Jadbabaie, A., J. Yu, and J. Hauser (2001), Unconstrained receding-horizon control of nonlinear systems, *IEEE Transactions on Automatic Control*, *46*(5), 776–783.

Johnson, S. G. (), The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

Karaman, S., and E. Frazzoli (2011), Sampling-based algorithms for optimal motion planning, *The International Journal of Robotics Research*, *30*(7), 846–894.

Karaman, S., and E. Frazzoli (2013), Sampling-based optimal motion planning for non-holonomic dynamical systems, in *2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 5041–5047.

Karaman, S., M. R. Walter, A. Perez, E. Frazzoli, and S. Teller (2011), Anytime motion planning using the RRT*, in *2011 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1478–1483.

Khalil, H. K. (2002), *Nonlinear Systems (3rd Edition)*, 3 ed., Prentice Hall, New York, NY, U.S.

Khatib, O. (1986), Real-time obstacle avoidance for manipulators and mobile robots, *The International Journal of Robotics Research*, *5*(1), 90–98.

Kikuuwe, R., N. Takesue, A. Sano, H. Mochiyama, and H. Fukimoto (2005), Fixed-step friction simulation: From classical coulomb model to modern continuous models, in *2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 1009–1016.

Kim, B., and P. Tsiotras (2002), Controllers for unicycle-type wheeled robots: Theoretical results and experimental validation, *IEEE Transaction on Robotics and Automation*, *18*(3), 294–307.

Knepper, R. A., and M. T. Mason (2009), Path diversity is only part of the problem, in *2009 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 3224–3229.

Knepper, R. A., and M. T. Mason (2012), Real-time informed path sampling for motion planning search, *The International Journal of Robotics Research*, *31*(11), 1231–1250.

Konolige, K. (2000), A gradient method for realtime robot control, in *2000 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 1, pp. 639 –646.

Konolige, K., M. Agrawal, R. Bolles, C. Cowan, M. Fischler, and B. Gerkey (2008), Outdoor mapping and navigation using stereo vision, in *Experimental Robotics*, *Springer Tracts in Advanced Robotics*, vol. 39, pp. 179–190, Springer Berlin Heidelberg.

Koren, Y., and J. Borenstein (1991), Potential field methods and their inherent limitations for mobile robot navigation, in *1991 IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 2, pp. 1398 –1404.

Kuipers, B., and K. Astrom (1994), The composition and validation of heterogeneous control laws, *Automatica*, *30*(2), 233–249.

Lam, C.-P., C.-T. Chou, K.-H. Chiang, and L.-C. Fu (2011), Human-centered robot navigation –towards a harmoniously human-robot coexisting environment, *IEEE Transactions on Robotics*, *27*(1), 99–112.

Lambert, A., D. Gruyer, and G. S. Pierre (2008), A fast monte carlo algorithm for collision probability estimation, in *10th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, pp. 406–411.

Lapierre, L., D. Soetanto, and A. Pascoal (2006), Nonsingular path following control of a unicycle in the presence of parametric modeling uncertainties, *International Journal of Robust Nonlinear Control*, *16*(10), 485–503.

LaValle, S. M. (1998), Rapidly-exploring Random Trees: A new tool for path planning, *Tech. Rep. 98-11*, Computer Science Dept., Iowa State University.

LaValle, S. M. (2006), *Planning Algorithms*, Cambridge University Press, Cambridge, U.K., available at http://planning.cs.uiuc.edu/.

LaValle, S. M. (2011a), Motion planning: The essentials, *IEEE Robotics and Automation Society Magazine*, *18*(1), 79–89.

LaValle, S. M. (2011b), Motion planning: Wild frontiers, *IEEE Robotics and Automation Society Magazine*, *18*(2), 108–118.

LaValle, S. M., and J. J. Kuffner (2000), Rapidly-exploring Random Trees: Progress and prospects, in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308.

Lee, J. H. (2011), Model predictive control: Review of the three decades of development, *International Journal of Control, Automation and Systems*, *9*(3), 415–424.

Lindemann, S. R., and S. M. LaValle (2009), Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions, *The International Journal of Robotics Research*, *28*(5), 600–621.

Luders, B., M. Kothari, and J. P. How (2010), Chance constrained RRT for probabilistic robustness to environmental uncertainty, in *AIAA guidance, navigation, and control conference (GNC)*.

Ma, X., C. Hu, X. Dai, and K. Qian (2008), Sensor integration for person tracking and following with mobile robot, in *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 3254–3259.

Magid, E., D. Keren, E. Rivlin, and I. Yavneh (2006), Spline-based robot navigation, in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 2296–2301.

Mason, M. T., and J. K. Salisbury Jr (1985), *Robot hands and the mechanics of manipulation*, The MIT Press, Cambridge, MA.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert (2000), Constrained model predictive control: Stability and optimality, *Automatica*, *36*(6), 789–814.

Micaelli, A., and C. Samson (1993), Trajectory tracking for unicycle-type and two-steering wheels mobile robots, *Tech. Rep. 2097*.

MicroStrain (), 3DM-GX2, http://www.microstrain.com/3dm-gx2.aspx.

Miller, D. P. (1996), Moving in tandem: automated person pacing for wheelchair users, in *Developing Assistive Technology for People with Disabilities*, AAAI-96 Fall Symposium Series, pp. 86–88.

Modayil, J., and B. Kuipers (2008), The initial development of object knowledge by a learning robot, *Robots and Autonomous Systems*, *56*(11), 879–890.

Nagarajan, U., G. Kantor, and R. Hollis (2013), Integrated motion planning and control for graceful balancing mobile robots, *The International Journal of Robotics Research*, *32*(1), 1005–1029.

Nakamura, K., H. Zhao, R. Shibasaki, K. Sakamoto, T. Ohga, and N. Suzukawa (2006), Tracking pedestrians using multiple single-row laser range scanners and its reliability evaluation, in *Systems and Computers in Japan*, pp. 1–11.

Oftadeh, R., R. Ghabcheloo, and J. Mattila (2015), A time-optimal bounded velocity path-following controller for generic wheeled mobile robots, in *2015 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 676–683.

Ogren, P., and N. Leonard (2005), A convergent dynamic window approach to obstacle avoidance, *IEEE Transactions on Robotics*, *21*(2), 188–195.

Palmieri, L., and K. O. Arras (2014), Distance metric learning for RRT-based motion planning for wheeled mobile robots, in *2014 IROS Machine Learning in Planning and Control of Robot Motion Workshop*.

Park, J. (2014), Modeling, state estimation, and steady-state control for a powered wheelchair, *Tech. rep.*

Park, J., and B. Kuipers (2011), A smooth control law for graceful motion of differential wheeled mobile robots in 2d environment, in *2011 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 4896–4902.

Park, J., and B. Kuipers (2013), Autonomous person pacing and following with Model Predictive Equilibrium Point control, in *2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1060–1067.

Park, J., and B. Kuipers (2015), Feedback motion planning via non-holonomic RRT*, in *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4035–4040.

Park, J., C. Johnson, and B. Kuipers (2012a), Robot navigation with Model Predictive Equilibrium Point Control, in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4945–4952.

Park, J., C. Johnson, and B. Kuipers (2012b), Robot navigation with MPEPC in dynamic and uncertain environments: From theory to practice, in *IROS 2012 Workshop on Progress, Challenges and Future Perspectives in Navigation and Manipulation Assistance for Robotic Wheelchairs*.

Perez, A., R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez (2012), LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics, in *2012 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2537–2542.

Piazzi, A., and C. Bianco (2004), Quintic g^2-splines for trajectory planning of autonomous vehicles, in *IEEE Intelligent Vehicle Symposium*, pp. 620–625.

Prassler, E., D. Bank, and B. Kluge (2002), Motion coordination between a human and a mobile robot, in *2002 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 1228–1233.

Rauskolb, F. W., et al. (2008), Caroline: An autonomously driving vehicle for urban environments, *Journal of Field Robotics*, *25*(9), 674–724.

Rawlings, J. (2000), Tutorial overview of model predictive control, *IEEE Control Systems*, *20*(3), 38 –52.

Rimon, E., and D. E. Koditschek (1992), Exact robot navigation using artificial potential functions, *IEEE Transactions on Robotics and Automation*, *8*(5), 501–518.

Satake, J., and J. Miura (2009), Robust stereo-based person detection and tracking for a person following robot, in *2009 ICRA Workshop on Person Detection and Tracking*.

Schouwenaars, T., J. How, and E. Feron (2004), Receding horizon path planning with implicit safety guarantees, in *American Control Conference*, vol. 6, pp. 5576–5581.

Singh, L., and J. Fuller (2001), Trajectory generation for a UAV in urban terrain, using nonlinear mpc, in *American Control Conference*, vol. 3, pp. 2301–2308.

Snape, J., J. van den Berg, S. Guy, and D. Manocha (2011), The hybrid reciprocal velocity obstacle, *IEEE Transactions on Robotics*, *27*(4), 696–706.

Takemura, H., N. Zentaro, and H. Mizoguchi (2009), Development of vision based person following module for mobile robots in/out door environment, in *2009 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pp. 1675–1680.

Tedrake, R., I. R. Manchester, M. Tobenkin, and J. W. Roberts (2010), LQR-trees: Feedback motion planning via sums-of-squares verification, *The International Journal of Robotics Research*, *29*(8), 1038–1052.

Thrun, S., W. Burgard, and D. Fox (2005), *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, The MIT Press, Cambridge, MA.

Topp, E., and H. Christensen (2005), Tracking for following and passing persons, in *2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 2321–2327.

Trautman, P., J. Ma, R. Murray, and A. Krause (2013), Robot navigation in dense human crowds: the case for cooperation, in *2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2153–2160.

Trautman, P., J. Ma, R. M. Murray, and A. Krause (2015), Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation, *The International Journal of Robotics Research*, *34*(3), 335–356.

van den Berg, J., M. Lin, and D. Manocha (2008), Reciprocal velocity obstacles for real-time multi-agent navigation, in *2008 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1928 –1935.

Von Hundelshausen, F., M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche (2008), Driving with tentacles: Integral structures for sensing and motion, *Journal of Field Robotics*, *25*(9), 640–673.

Walters, M., K. Dautenhahn, R. te Boekhorst, K. L. Koay, C. Kaouri, S. Woods, C. Nehaniv, D. Lee, and I. Werry (2005), The influence of subjects' personality traits on personal spatial zones in a human-robot interaction experiment, in *IEEE Int. Workshop on Robot and Human Interactive Communication (ROMAN)*, pp. 347–352.

Wang, C. M. (1988), Location estimation and uncertainty analysis for mobile robots, in *1988 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1231–1235.

Webb, D. J., and J. v. d. Berg (2012), Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints, *arXiv preprint arXiv:1205.5088*.

Weisstein, E. W. (), Archimedean spiral, from *MathWorld–A Wolfram Web Resource*, http://mathworld.wolfram.com/ArchimedeanSpiral.html.

Zhang, L., S. M. LaValle, and D. Manocha (2009), Global vector field computation for feedback motion planning, in *2009 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 477–482.