MOTION PLANNING AND COORDINATION OF MOBILE ROBOT BEHAVIOR FOR MEDIUM SCALE DISTRIBUTED WIRELESS NETWORK EXPERIMENTS

by

Daniel Montrallo Flickinger

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

The University of Utah

December 2006

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Daniel Montrallo Flickinger

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

	Chair: Mark A. Minor
	Sanford G. Meek
	Frank J. Lepreau

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Daniel Montrallo Flickinger in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Mark A. Minor
Chair, Supervisory Committee

Approved for the Major Department

Kent S. Udell
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

In this research, a computerized motion planning and control system for multiple robots is presented. Medium scale wheeled mobile robot couriers move wireless antennae within a semi-controlled environment. The systems described in this work are integrated as components within Emulab Mobile, a wireless research testbed. This testbed is publicly available to users remotely via the Internet. Experimenters use a computer interface to specify desired paths and configurations for multiple robots. The robot control and coordination system autonomously creates complex movements and behaviors from high level instructions.

Multiple trajectory types may be created by Emulab Mobile. Baseline paths are comprised of line segments connecting waypoints, which require robots to stop and pivot between each segment. Filleted Circular arcs between line segments allow constant motion trajectories. To avoid curvature discontinuities inherent in line-arc segmented paths, higher order continuous polynomial spirals and splines are constructed in place of the constant radius arcs.

Polar form nonlinear state feedback controllers executing on a computer system connected to the robots over a wireless network accomplish posture stabilization, path following and trajectory tracking control. State feedback is provided by an overhead camera based visual localization system integrated into the testbed. Kinematic control is used to generate velocity commands sent to wheel velocity servo loop controllers built into the robots.

Obstacle avoidance in Emulab Mobile is accomplished through visibility graph methods. The Virtualized Phase Portrait Method is presented as an alternative. A virtual velocity field overlay is created from workspace obstacle zone data. Global stability to a single equilibrium point, with local instability in proximity to obstacle regions is designed into this system.

The design, implementation, integration and analysis of these systems is presented in this research. Experiments are completed to evaluate the performance of motion planning and control under real world conditions.

To Mable

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Thanks to Mark Minor, Sanford Meek, and the Flux Research Group. (Especially David Johnson, Tim Stack, Russ Fish, Leigh Stoller, and Jay Lepreau.) Thanks to Youngshik Kim for his work on the kinematic controller.

CHAPTER 1

INTRODUCTION

The design and implementation of a multiple robot motion control system for a wireless networking research testbed using medium scale robots as couriers is presented in this research. Major aspects of this system, including trajectory generation, motion control, obstacle avoidance, and multiple robot coordination are discussed. In addition to the implementation of this system in a real world environment, the challenges related to localization, communications, and control are analyzed.

It is hypothesized that remote kinematic state feedback control of multiple robots simultaneously, with visual localization running at 30 Hz is feasible, and capable of trajectory tracking with minimal error. This research aims to establish this, and covers all the design and engineering decisions involved in creating a system capable of evaluating robot motion control under these constraints.

1.1 Emulab Mobile

Emulab mobile [1], an extension to Emulab [2], is designed to allow remote users to interactively conduct wireless network experiments. Medium scale robots are used as couriers to position equipment such as antennae and computing hardware within a semicontrolled environment.

Wireless network simulations may produce inadequate results. There exist effects which are difficult to model, resulting in differing results when comparing simulation to experimental data. Emulab Mobile is intended to provide the infrastructure required to obtain data from real world experimentation, using mass produced, commercially available wireless networking devices. This allows research groups access to remotely conduct experiments, without the initial cost and effort associated with the implementation of a wireless network testbed.

Emulab and Emulab Mobile are remotely accessible to experimenters over the Internet. Emulab provides a web-based experimentation interface for users to create, modify and

manage network experiments. A Java applet interface is included in Emulab Mobile to visualize and coordinate control of multiple robots simultaneously while conducting experiments.

1.2 Requirements and Goals

Robots are chosen for Emulab mobile based on capabilities such as speed, battery life, and payload. Factors such as cost, maintenance, ease of use, and commercial support and availability are also strongly considered. Commercially available robots are desired, in order to speed development time, and build a complete system that can be reproduced by other research groups.

The robot workspace is best described as a standard *office space* type environment. It consists of a flat carpeted area of 60 meters squared. Furnature such as chairs, tables, shelves, a couch, and other miscellanea are present around the perimeter of the area. The environment, termed as semicontrolled, has frequent foot traffic, causing transient obstacles to be present in the workspace. Robots capable of operating in this environment are required to fulfill courier duties in Emulab Mobile.

Robots must be capable of untethered, autonomous operation. Remote usage is an important feature of Emulab Mobile. Minimal operator support is desired in order to maximize the utility of the robots as couriers. Commands delivered over a wireless ethernet control network are required, and the robots must be able to carry the hardware required to achieve this.

It is desired that robots operate at near their rated maximum velocity of 2.0 meters per second. Reconfiguration of multiple robots in the workspace must be completed as fast as possible in order to reduce the time required to run experiments. Path tracking error must be as low as possible, and goal positioning needs to be as accurate as 1.0 centimeters.

1.3 Constraints

1.3.1 Robots

The robot couriers chosen are six Acroname Garcia robots. Chosen for their cost and capabilities, these medium scale robots are used to move the equipment needed for wireless network experimentation within Emulab Mobile. The Garcia is a differentially steered wheeled mobile robot propelled by electric motors, powered by a rechargable battery.

1.3.2 Localization

Cost and availability constraints dictate that an overhead camera localization system is used on Emulab Mobile. This system consists of six overhead color security cameras pushing data to a custom built system designed to detect fiducial markers placed on top of each robot. The system provides continual localization data to the robot control system in the form of a global Cartesian position, plus an orientation for each robot in the workspace. To keep costs low, the cameras used are only capable of 30 Hz framerates, which in turn limits the availability of localization data to 30 Hz intervals. This localization data is used as state feedback for a motion controller.

1.3.3 Software Systems

Emulab Mobile is closely tied to Emulab, which provides an established software base for building the robot control system. A user interface, event system, and other supporting software are included outside the scope of this research. Applications to coordinate communications, obtain localization data, and send wheel velocity commands to robots where created by researchers at the Flux Research Group, and are only discussed in this research to provide background information. All components of this research are incorporated as part of the robot control application, termed RMCD, of which an overview is given in Subsection 2.5.2.

More details about the various software components of Emulab Mobile are discussed in Section 2.5. These components accomplish all coordination, communication, and control of robots in the testbed system.

1.4 Challenges

The constraints presented in Section 1.3 create several challenges, which are addressed in this research. The operating environment, software environment, robot hardware, and cost constraints all create problems that must be solved. The system presented in this research address these issues and provide a working system that allows robots to operate as couriers in a semicontrolled environment.

1.4.1 Environment

The environment in which robots must operate within Emulab Mobile is different from many of the more controlled lab environments used in other research. Robots must avoid interaction with static obstacles, but additionally must account for transient obstacles

and multiple robot interaction issues. The carpeted surface present in the testbed area is further detrimental to localization through odometry due to wheel slip.

1.4.2 Robot Performance

The robot cost and commercial availability constraints place challenges on the overall capabilities of the robots. The robots chosen for use on the testbed are advertised to have a maximum velocity of two meters per second. In reality, the maximum velocity observed is much lower. This is due to the fact that the robots are operating on carpet, and also because of the size of the workspace. There is not enough space for a robot to accelerate to maximum velocity before crossing a boundary or encountering an obstacle.

1.4.3 Communications

All localization data, user commands, and robot control commands are passed over an Ethernet network, with any communication between the central control system and the robots taking place over wireless Ethernet. In the implementation on Emulab Mobile, the motion controller is run within the central control system, with the robots receiving only wheel speed commands over the wireless link. When the link intermittently fails because of interference or other factors, the main control loop is broken. The use of this link also presents problems with latency.

1.4.4 Trajectory Specification

Trajectories must be specified by users who may not be familiar with the field of robotics. No mention of nonholonomic or other kinematic constraints is given within the user interface for Emulab Mobile.

1.4.5 Robot Control

Motion control of multiple robots with a kinematic state feedback controller running at a 30 Hz sampling rate over a potentially lossy communications link is the biggest challenge in this research. In an effort to save vital on board computational resources, the main motion controller can not be run on board the robots. Localization data can not be provided at a rate any faster than 30 Hz, limiting the rate at which the control loop can be executed.

1.4.6 Multiple Robot Coordination

The coordination and control of multiple robots simultaneously presents several challenges. Most significant is the prospect of robot collisions, which can cause severe deviation from prescribed trajectories. Deadlocking is another issue, where multiple robots mutually block each other from achieving their objectives.

1.5 Contributions

The main contribution of this research is the remote autonomous control of mobile robots with insufficiently fast control systems over lossy communication channels. Kinematic trajectory tracking of parametric reference trajectories is accomplished on multiple robots under wireless control with control loops running at 30 Hz. Localization is achieved through visual tracking only, with minimal localization capabilities present on individual robots.

This research encompasses the engineering challenges encountered while designing and implementing a system providing motion control and coordination of multiple robots. Robots are used for tasks in the real world, and controlled by high level commands given by researchers not necessarily familiar with robotics. The motion control system is part of a much larger, separate body of work used for network research, called Emulab. Emulab provides features useful for the creation of experiments, along with the required user interface facilities, computing and networking hardware, and organizational structure. The topics presented in this research are part of a greater contribution to provide mobile wireless networking experimentation capabilities to Emulab.

1.6 Document Structure

The remaining portions of this document are structured as follows: Chapter 2 contains background material concerning trajectory generation, motion control, and obstacle avoidance, along with some background about the Emulab Mobile testbed system. The main research topics follow in the next few chapters, with trajectory generation discussed in Chapter 3, motion control in Chapter 4, and finally, obstacle avoidance in Chapter 5.

The design of simulations for the above systems and associated results are presented in Chapter 6. Discussion of implementation issues for the various trajectory generators and motion controllers is given in Chapter 7. Experimental results concerning these areas are given in Chapter 8, and concluding remarks can be found in Chapter 10.

CHAPTER 2

BACKGROUND

Other wireless networking research testbed exist, but none with robot couriers capable of unconstrained motion within a similar area. Many testbeds offer mobility of devices, but only through tethered robots with small workspaces. Internet based control has been offered on other systems to control the motion of robots. The Emulab Mobile testbed integrates commercial hardware into a full research testbed, at minimal cost, and available to anyone for use from any location.

The usage of line and circular arc based paths is well established in the research, as is the generation of more continuous curvature arcs. Trajectory generation is discussed in detail in Section 2.1.

Motion control through state feedback running on remote systems is novel; as is the control law applied to this system to simultaneously solve the posture regulation, path following, and trajectory tracking problems. New research is accomplished into discrete stability analysis of controllers operating at slow sampling rates. More details about current motion control methods utilizing state feedback are discussed in Section 2.2.

A novel obstacle avoidance method similar to artificial potential field methods is presented in this research. Other obstacle avoidance methods are discussed in Section 2.3. A background of the system hardware and software architecture is given in Section 2.4 and Section 2.5.

2.1 Trajectory Specification

A feasible reference goal posture, path, or trajectory is required before a motion controller can move a robot. In its most basic form, a goal posture is set, and a robot is turned to orient towards the goal, then driven in a straight line to the goal, then finally oriented to the goal posture.

Wheeled robots are capable of more complex motions, and only a subset of robots are actually capable of reorienting by pivoting through a zero radius turn about their

center. Turns of arbitrary radius are possible, and paths must be generated to take advantage of this. Segmented paths can be constructed. Segments may involve motion in a straight line, or a turn. Path specification progresses in complexity to the point of full parametric trajectories defining instantaneous posture, and its derivatives, resulting in smooth, complex, continuous motion along a defined path, at defined velocities and accelerations.

Nonholonomically feasible path generation can be accomplished by smoothing non-feasible paths. These generators take polygonal waypoint based paths, and insert curves to produce more continuous paths. This type of generator forms a path that is feasible considering the nonholonomic constraint [3].

The geometric curvature continuity of curves used in path generation is an important aspect of the design of feasible paths [4]. Line and circular arc based paths, pioneered by Reeds and Shepp, have discontinuities in curvature at the points between lines and arcs. The curvature of a straight line segment is zero, and the curvature instantaneously increases to a nonzero number when transitioning from a line to an arc segment. Line and circular arc segment paths can be extended to have more continuous curvature [5].

There exist two main categories of curves, closed form, such as circular arcs and splines, and parametric curves with curvature as a function of arc length. Closed form circular arcs are defined by their curvature, from which a radius can be calculated. To arrive at a solution for the curve, a center point, radius, and boundary points are needed. Parameterization of the curve can be done geometrically. Splines are more complex, but still may be solved with geometric methods and straight forward construction.

Parametric curves specified by curvature as a function of arc length allow direct control over curvature continuity. They are more difficult to construct with boundary constraints at each end of the curve.

Clothoids are well known curves for smoothing paths [6] [7], and have been heavily utilized in roadway and railroad design. Two Clothoids are symmetrically paired, with zero curvature at the boundaries, and maximum curvature at the center of the arc, at the intersection of the two Clothoids. A major problem with Clothoids, and polynomial spirals in general is the lack of a closed form solution. Fresnel integrals must be solved to obtain the coefficients required to meet specific boundary conditions in Cartesian space. Parametric curves can be created from polynomial spirals of any order [8]. Cubic spiral curves [9] are analogous to Clothoid pairs. Optimal control theory may be used to solve for

smooth curvature and derivatives of curvature, while minimizing the maximum curvature [10]. This produces a smooth curve that can be tracked by a nonholonomic robot with minimal wheel slippage and tracking error, allowing higher velocities.

2.2 Motion Control

There are four major classes of motion control considered in this research. As a baseline, point to point motion comprised of straight line motions interrupted by pivots, tracking polygonal waypoint based paths is considered. Posture stabilization may be used to execute these polygonal paths, or to stabilize a robot to a single final goal posture. To reduce motion time and expedite robot arrival to final goal points, more continuous paths are used, and more advanced motion controllers are needed. This class of motion controller is capable stabilizing a robot to a parametric path, or a parametric trajectory.

Two forms of controllers may be used for motion control of wheeled mobile robots. Kinematic control involves stabilizing a kinematic system by controlling a robot through velocity inputs. Dynamic control includes a system model including dynamic and kinematic states, and generally uses torque commands as inputs to the robot. A hybrid form of control can be used, in which kinematic and dynamic control are combined through the use of a dynamic extension [11]. Feedback stabilization of the nonholonomic system is used to solve the goal stabilization, path tracking, and trajectory tracking problems [12].

2.2.1 Posture Stabilization

Posture stabilization of wheeled mobile robots may be accomplished using a kinematic state feedback linearizing controller [13]. This controller is employed for posture regulation only, but can be extended to perform path following. A unicycle kinematics model is specified, which is applicable to a differentially-steered vehicle.

A similar technique is applied to compliant frame robots, formed from two coupled differentially steered robots, also using Lyapunov analysis [14]. This vehicle has additional constraints over a vehicle with unicycle kinematics, such as a minimum radius of curvature. The time invariant smooth control law developed for this application employs a minimum radius of curvature, and also limits the vehicle to forward motion only.

The Lyapunov technique can also be used for a vehicle with bicycle-like kinematics. [15]. Cartesian state equations are converted into polar form, and a velocity and curvature

based control law is developed. As with the compliant frame vehicle, this control law limits the vehicle to forward motion only.

An example of path planning for obstacle avoidance is the use of a sliding mode controller with a potential field and obstacle exclusion zones based on electrical charge models. [16]. In this method, sliding mode control is used to direct a path around a known obstacles. Another sliding mode controller executes above the single obstacle path planner to plan paths with multiple obstacles by considering a single obstacle at each time increment. Various optimizations are made to eliminate large accelerations and tight turning radii.

2.2.2 Trajectory Tracking

Trajectory tracking control involves stabilizing a robot to a continuously moving reference frame. A trajectory has both position and velocity parameters, and may additionally include acceleration and curvature references. Trajectory tracking control can be accomplished with feedback from visual data [17]. Noisy image data can be filtered, and used to estimate robot kinematic state data. An Extended Kalman Filter may be used to provide robust state estimations based on noisy data from imaging systems.

Lyapunov design may be used to develop a smooth, time invariant control law to solve the trajectory tracking problem [18, 14]. The kinematic system must be modeled in Polar form, or a smooth, time invariant control law is not possible.

2.3 Obstacle Avoidance

The earliest example found in the literature of mobile robot obstacle avoidance using harmonic potential fields is the pioneering work by Khatib [19]. Utilizing this method, virtual attractive and repulsive forces acting upon manipulators and mobile robots are created by generating a dipolar field between the center of an obstacle, and the goal point. A path is then created by solving the gradient of the potential field.

Problems with potential field methods are well described in the literature [20]. The most common problem is the creation of traps due to local minima. Paths resulting from solving a potential field may experience lateral oscillations in the presence of multiple obstacles. This is especially a problem with methods that consider multiple obstacles by switching between single obstacles.

To diminish oscillations of paths resulting from potential functions, path generation using a two dimensional Gaussian function can be used [21]. Oscillations can be eliminated

using Newton's method, instead of a gradient method.

It is analytically difficult to design potential fields restricted to have a single minimum coinciding with the goal. This is especially difficult when considering multiple obstacles concurrently. The generation of local minima may be suppressed with potential functions using Laplace's equation [22]. Instead of suppressing the generation of local minima, multiple minima can be searched with a graph search algorithm to achieve a final goal configuration [23]. The major drawback with this method is the increased computation requirements, especially when considering dynamic obstacle environments.

Sliding mode control can be used to allow a nonholonomic wheeled mobile robot track reference paths generated through potential energy fields. A single field for each obstacle is modeled, while a higher level *spatial* controller is used to apply only the field from the obstacle closest to the robot. Local minima are not likely to be created due to the consideration of only a single obstacle at a time. An electrostatic field is used, guaranteeing that field lines do not escape to infinity [16]. A Coulomb model electrostatic potential field with harmonic Laplacian properties can be used to suppress local minima [24]. High path curvature can result without modifying the sliding surface by shrinking obstacle regions gradually when the robot comes in proximity.

The problem of local minima in potential field methods can be avoided by using geometric methods such as cellular decomposition, and Voronoi diagrams. With cellular decomposition, the free space is divided into discrete cells, which are heuristically searched using graph theory to obtain a partial ordering of cells starting from the goal point [25]. Voronoi diagrams can be applied to generate a safe reference path from initial to goal configurations. A major problem with this type of solution is that smooth continuous paths are not always created. Reference paths are generally polygonal when using geometric path planning methods, but may be smoothed to fit within the nonholonomic kinematic constraints of certain wheeled mobile robots.

VPPM avoids many of these issues related to potential fields by utilizing the properties of nonlinear system equations to control equilibrium points, and minimize oscillations. The VPPM field can be designed to provide a desired path or velocity profile, with highly adjustable field overlay parameters.

The obstacle field overlay can be modified in such a way that field vectors are directed outwards from the geometric center of a cluster of obstacles. An obstacle field is not constrained to act only radially outward from the center of its corresponding obstacle

Length	10.980 in	0.2789 m
Width	7.673 in	0.1949 m
Height (head)	3.680 in	0.0935 m
Track Width	7.000 in	0.1778 m
Wheel radius	3.996 in	0.1015 m

Table 2.1. Garcia robot specifications.

region. Given this, unidirectional obstacle fields can be created, allowing groups of obstacles to have custom-designed field polarities. This allows local minima creation to be eliminated.

Our method allows the field in proximity to obstacle regions to be statically tuned for desired path curvature. The field does not need to be reactively changed in relation to the current position of a robot.

2.4 Hardware

The core components of Emulab Mobile are the robots, localization system, computing hardware, and wireless networking hardware. These systems are all comprised of commercially available products, with no specialized hardware present in the system. Commercial hardware is chosen to limit costs, and to lower the barrier to implementation of Emulab Mobile by other research groups. An entire system can be bought and set up to a full working system by any moderately equipped group.

2.4.1 Robots

Six Acroname Garcia [26] robots are used as couriers by Emulab Mobile. This is a commercially available robotics platform. Specifications for the Garcia robot are found in Table 2.1. A photograph of two Garcia robots in their normal workspace is shown in Figure 2.1. The colored fiducials used for localization are visible in this photograph, as are the on board computers, and wireless antennae.

The Garcia robots are chosen as couriers in Emulab Mobile in the spirit of using commercially available hardware. The robots are available for purchase as a commercial product, and include all the components preassembled. Furthermore, a software API is provided that allows point to point motion, along with the ability to command wheel velocities directly.

The robots are of steel construction, with an electric motor power each of the two wheels independently. Power is obtained from rechargable nickel metal hydride batteries,

commonly used in hobby scale radio controlled vehicles. The robots each come configured with a Stargate computer system mounted within the case, at the top.

Slight modifications are performed to each robot. Colored fiducials are added to the top to allow tracking by the visual localization system. The mounting plate for the Stargate computer is raised to allow the inclusion of addition equipment needed for sensor network experimentation. Finally, a long antenna extension is added, to place the antenna close to human hip height. This antenna height is desirable to model mobile devices that may be carried by persons either in a pocket, or clipped to a belt.

2.4.2 Localization

Robot localization is accomplished by an overhead camera system installed in the robot workspace. Downward looking cameras continually track colored fiducial markings on each robot. A series of cameras output localization data used for robot tracking and

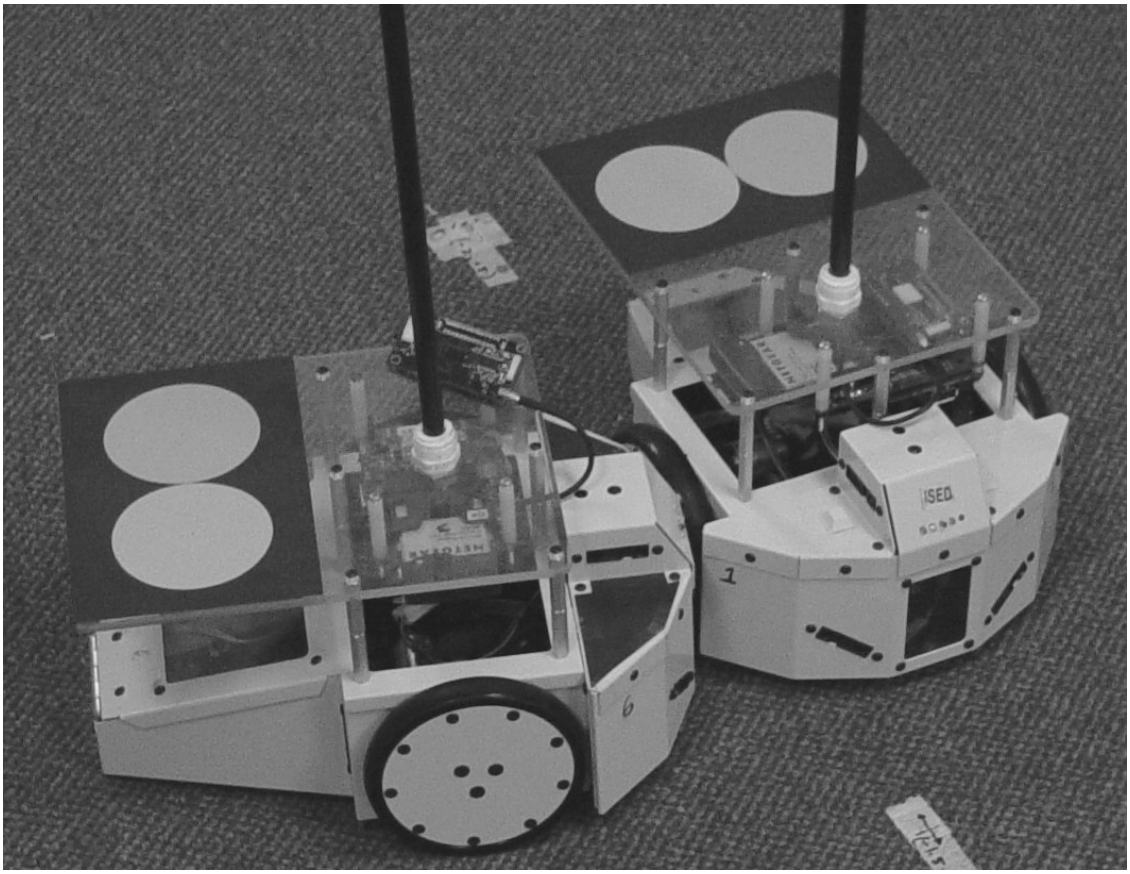

Figure 2.1. Two Garcia robots.

state feedback. Corrections are made for lens distortion, resulting in a mean position error estimate of approximately one centimeter.

2.5 System Architecture

Emulab Mobile is divided into four distinct software systems, termed daemons. These applications run noninteractively, and handle all the calculation, communication, and data collection needed to control robots in the testbed environment. The three main master control daemons run as single instances for each experiment in the background on a fixed computer, while instances of the Garcia Pilot application run on the local computer on board the robots. The master control daemons communicate over an Ethernet connection, while the Robot Master Control Daemon communicates with instances of Garcia Pilot over a wireless Ethernet link.

An overview of the Emulab Mobile system architecture is shown in Figure 2.2. The components which are part of Emulab Mobile are on the left side of the diagram, denoted by light gray boxes. The visual localization block is a part of VMCD, while the rest of the blocks belong to RMCD.

2.5.1 Emulab Master Control Daemon

The Emulab Master Control Daemon (EMCD) handles communication from Emulab, users and VMCD. Motion commands, reference waypoints, and posture updates from VMCD are forwarded to RMCD. Feedback from RMCD is collected, and passed to Emulab.

EMCD is the primary system to manage data, communicate with Emulab, and coordinate robots. All high level motion commands and localization data are passed through

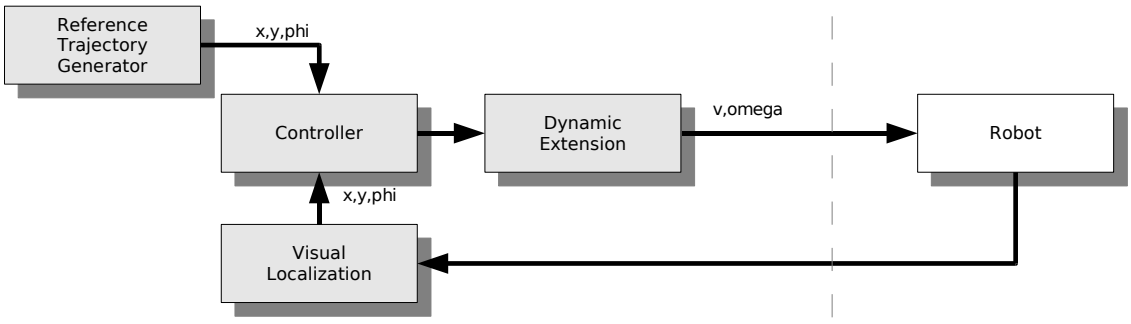

Figure 2.2. Emulab Mobile system architecture overview.

and coordinated within EMCD.

2.5.2 Robot Master Control Daemon

The Robot Master Control Daemon (RMCD) coordinates motion of multiple robots. It builds reference trajectories from waypoint data, and sets goal points for robots based on user input passed from EMCD. RMCD runs the state feedback controllers, and sends wheel speed commands to instances of Garcia Pilot.

2.5.3 Vision Master Control Daemon

The Vision Master Control Daemon (VMCD) collects and processes data from the overhead camera localization system. When robots are undergoing point-to-point motion, localization updates are sent to EMCD when requested. When state feedback control is active, localization data is continually sent to RMCD through EMCD.

2.5.4 Garcia Pilot

Instances of the Garcia Pilot application run on each robot. Under point-to-point motion, Pilot sends line and pivot motion commands, giving feedback based on odometry. While under state feedback control, Pilot passes through wheel speed commands sent from the controller running on RMCD. Pilot also handles sending back telemetry data to RMCD.

Garcia Pilot utilizes an API provided by the manufacturer of the robots. This API provides functions to execute movement commands, termed primitives, which use internal odometry to move or pivot a robot in a straight line by a prescribed distance or angle. Functions are also provided to set individual wheel speeds, using an internal PID servo loop.

The development new components to Emulab Mobile are discussed in the following three chapters. The design of motion planning and control systems is presented, leading to the simulation, testing, and verification of the system.

CHAPTER 3

TRAJECTORY GENERATOR

For every user specified goal point or baseline trajectory required for experimentation, a full robot trajectory must be created autonomously. In instances where only a single goal point is desired for each individual robot, an iterative line segment based trajectory generator with obstacle avoidance capabilities is executed, as presented in Section 3.1. In other situations, where a specific trajectory is desired for each robot, user specified waypoints are collected, and the resulting connecting line segments are filleted to create trajectories consisting of alternating lines and arcs. This waypoint model is discussed in Section 3.2. Different types of curves are placed in the fillets in accordance with the curvature continuity, velocity, and timing requirements of the robots.

A number of curve types may be chosen, based on the usage of line segments and filleted arcs. Constant radius circular arcs, discussed in Section 3.3, are used as the baseline curve type for their path length optimality, and straightforward geometric construction. To improve continuity at segment boundary points, polynomial spirals may be used in the fillets between segments. These spirals, discussed in Section 3.4, are described in terms of curvature versus arc length.

Splines are used to obviate some of the problems inherent in polynomial spirals. Parameters are in Cartesian space, instead of functions of arc length and curvature, decreasing the compution required to achieve boundary conditions. Quintic splines are discussed in detail in Section 3.5. This class of curve is well established in the computer graphics and computer aided design fields.

For all curves, there are tradeoffs with curvature constraints. A lower curvature derivative at the boundary of a segment results in higher maximum curvature. A higher maximum curvature reduces the maximmum allowable velocity along the curve, as limited by attainable traction forces and other maximum force limits. A higher curvature derivative at the curve boundaries increases maximum acceleration, and causes tracking error when robot acceleration limits are reached.

In this chapter, two different motion models are presented. The model in Section 3.1 is based upon autonomous path generation from line segments, while the rest of the sections in this chapter present motion models derived from Reed-Shepp paths.

3.1 Iterative Goal Point Progression Waypoint Model

The robot workspace is modeled as a series of rectangular zones. Areas covered by the localization system are considered as safe zones, while areas containing obstacles are exclusion zones. Using this model, a waypoint-based iterative path planner is employed to send robots to user selected destinations. To create a path for a robot, a user selects a single goal point. The robot is driven iteratively closer to this goal point using a modified visibility graph algorithm.

This waypoint based motion model relies on straight line and pivot motions, achieved by using motion commands, termed primitives, built in to the robots. These primitives require only a linear distance, or angular measurement arguement, and use odometry to complete the commands. A pivot and a linear displacement command are combined to form a meta command, called the goto command. The goto command uses an initial pivot followed by a linear move to send a robot to any arbitrary Cartesian relative position.

An example path created through a sparse obstacle field is given in Figure 3.1. An exclusion zone exists around the obstacle, expanding 0.25 meters from the obstacle boundary. The width of the expansion is chosen to be greater than the maximum robot dimension radially drawn from the center point of the robot. Corner points are created on these exclusion zones, and are used as intermediate robot waypoints.

Figure 3.2 illustrates the steps required for a robot to negotiate an obstacle. A robot is represented as a triangle, with dashed lines denoting the path direct to the goal point. The solid line in Step 2 is the resulting intermediate path created to the nearest obstacle exclusion zone corner point. The goal is represented as a star. As follows are the steps taken by the iterative path generator.

Step 1 Create a line segment with endpoints at the current robot position and the final goal position chosen by the user.

Step 2 If the current line segment does not intersect any obstacle exclusion zones, the intermediate goal position is set to the final goal position.

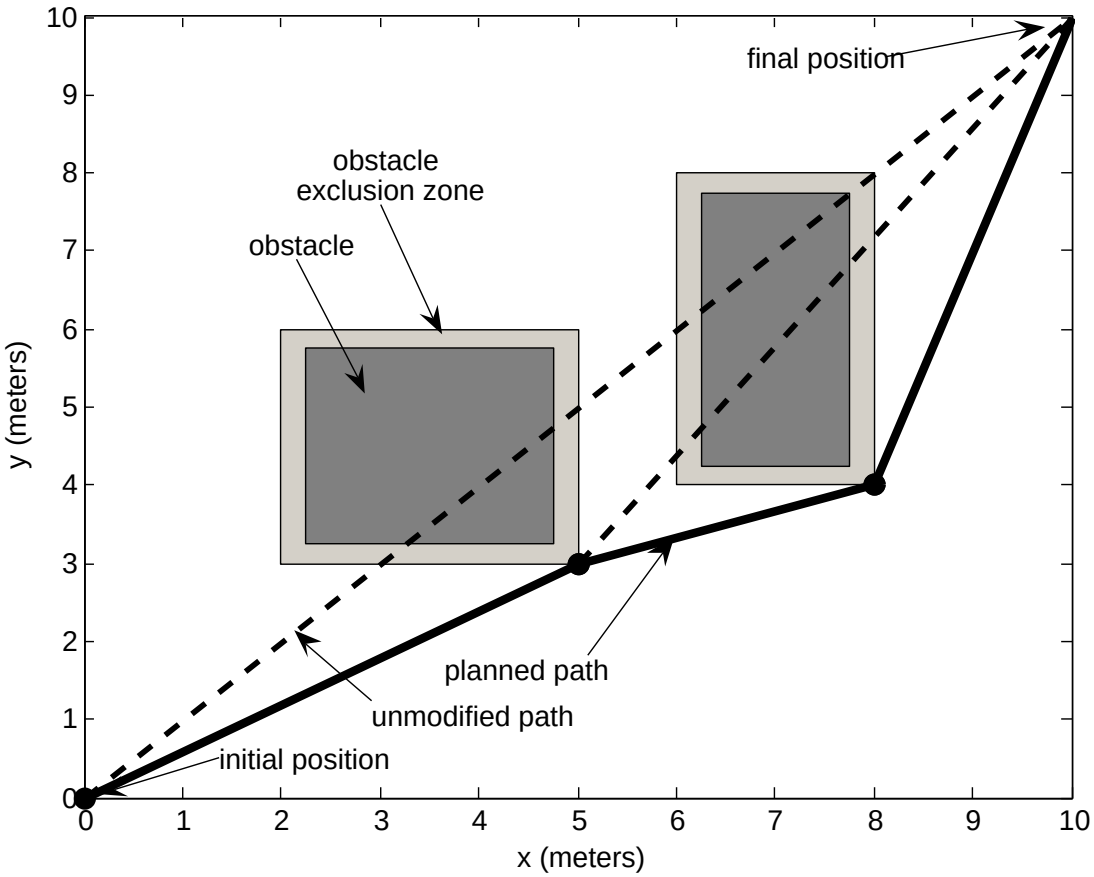

Figure 3.1. Path generation using iterative goal point progression method.

If the current line intersects an obstacle exclusion zone, the zone corner point closest to both the current position, and closest to the goal point is chosen as the intermediate goal position.

Step 3 Drive the robot to the intermediate goal position.

Step 4 If the robot is not at the final goal position, return to the first step.

The iterative waypoint method possess several drawbacks. The most significant of which is the high elapsed time between motion start and arrival at the goal point. The requirement that robots pivot at each waypoint wastes time, and limits the maximum velocity attainable. There is no support for user-specified paths, only singular goal points. The method itself is limited in its scope and implementation, and only appropriate for a workspace modeled by rectangular regions.

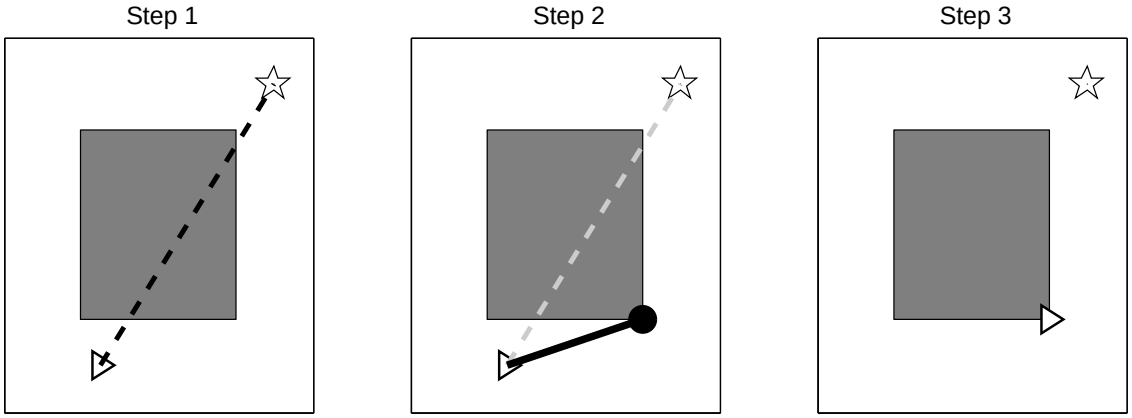

Figure 3.2. Path generation steps.

3.2 User-specified Waypoint Model

To allow faster movements, and more complex trajectory specifications, a second waypoint based motion model is created. Users choose multiple waypoints to specify a path for a robot, instead of choosing only a single goal point. Waypoints are connected by straight line segments comprising a path from a start to a goal position. Instead of commanding robots to pivot at each waypoint, the path segments are filleted, with curve segments inserted between straight line segments. The resulting path of line segments and arcs allows robots continuous motion to workspace destinations. Different types of arcs can be used to build a path between the filleted segments, with varying curvature continuity properties.

Two different solutions to calculate reference trajectories may be used. These methods are compared in Figure 3.3. The kinematics based method uses the Cartesian robot state equation, (4.1), to calculate x, y, and ϕ given reference velocities v and ω. Discontinuities in curvature at the interfaces between line and circular arc segments cause drift to occur. A closed-form solution to parameterizing reference trajectories is desired to eliminate this drift.

3.2.1 Line Segment Filleting

The radius of curvature is determined by specifying a reference velocity. Higher reference velocities require larger radius curves to prevent the loss of wheel traction. A fast, closed form geometric solution is needed to create arcs for trajectory generation. Presented here is a method utilizing intersecting offset line segments to solve for the

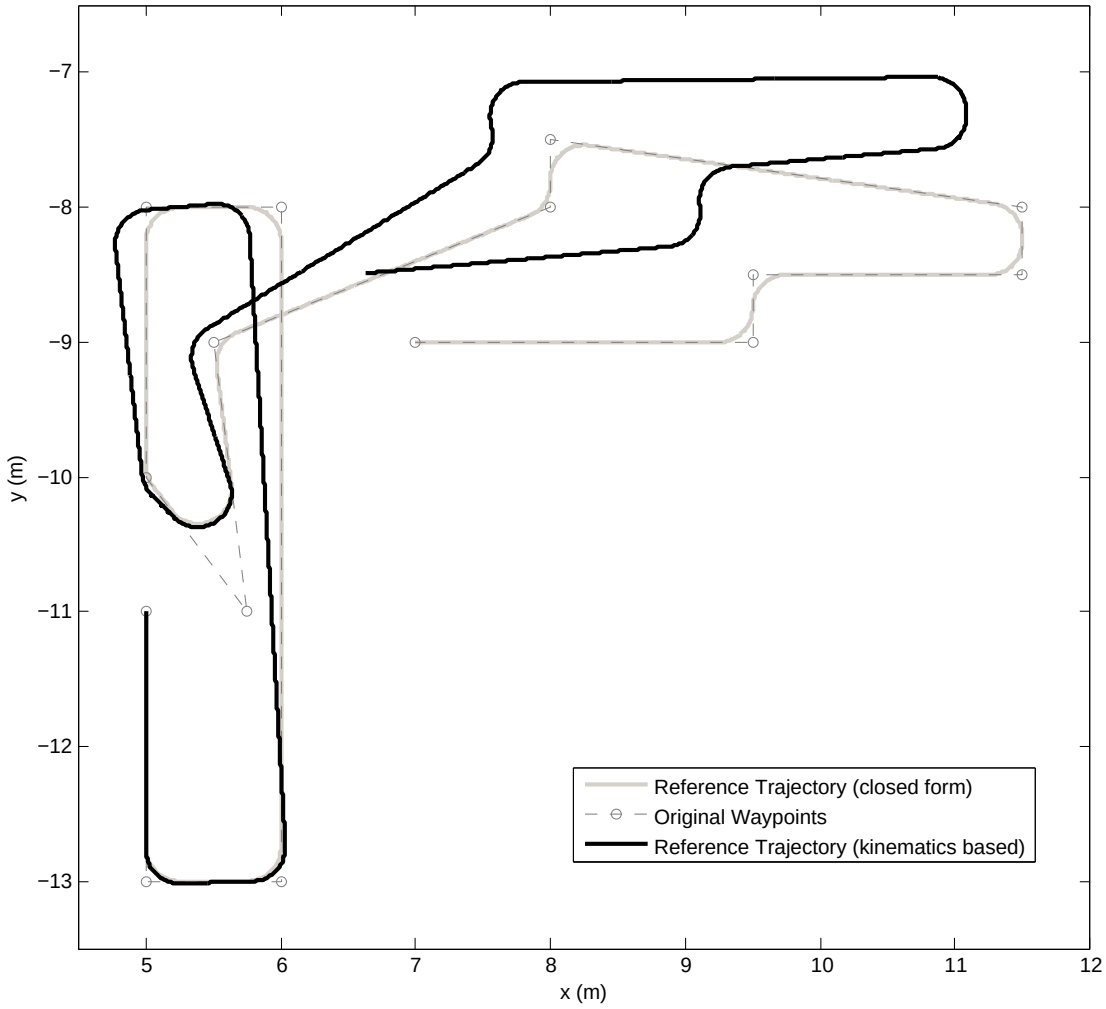

Figure 3.3. Comparison of kinematic and closed form trajectory generators.

center of a circular arc.

Given three input waypoints a, b, and c, as illustrated in Figure 3.4 and Figure 3.5, line segments $\vec{ab}$ and $\vec{bc}$ are constructed. These segments are represented by dotted lines. Two lines, denoted as dashed lines in these figures, represent lines offset by distance r from segments $\vec{ab}$ and $\vec{bc}$. The intersection of these offset lines corresponds with the arc center point, f.

Two unit vectors, perpendicular to the waypoint line segments are calculated,

$$\|\vec{df}\| = sign(\vec{ab} \times \vec{bc}) \cdot R_z\left(\frac{\pi}{2}\right) \cdot \vec{ab}/\|\vec{ab}\|, \tag{3.1}$$

and

$$\|\vec{ef}\| = sign(\vec{ab} \times \vec{bc}) \cdot R_z\left(\frac{\pi}{2}\right) \cdot \vec{bc}/\|\vec{bc}\|. \tag{3.2}$$

Where $R_z\left(\frac{\pi}{2}\right)$ is a rotation about the z axis by $\frac{\pi}{2}$; defined by

$$R_z\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{3.3}$$

Points are constructed to build line segments offset to the original line segments,

$$p_1 = a + r \cdot \|\vec{df}\|, \tag{3.4}$$

$$p_2 = b + r \cdot \|\vec{df}\|, \tag{3.5}$$

$$p_3 = b + r \cdot \|\vec{ef}\|, \tag{3.6}$$

$$p_4 = c + r \cdot \|\vec{ef}\|. \tag{3.7}$$

Where r is the desired arc radius, corresponding to the offset distance.

The intersection of the two offset line segments is determined by calculating the determinants of the points, such that,

$$L_1 = \left| \begin{bmatrix} {p_1}^T \\ {p_2}^T \end{bmatrix} \right|, \tag{3.8}$$

and

$$L_2 = \left| \begin{bmatrix} {p_3}^T \\ {p_4}^T \end{bmatrix} \right|, \tag{3.9}$$

and

$$M = \left| \begin{bmatrix} (p_1 - p_2)^T \\ (p_3 - p_4)^T \end{bmatrix} \right|. \tag{3.10}$$

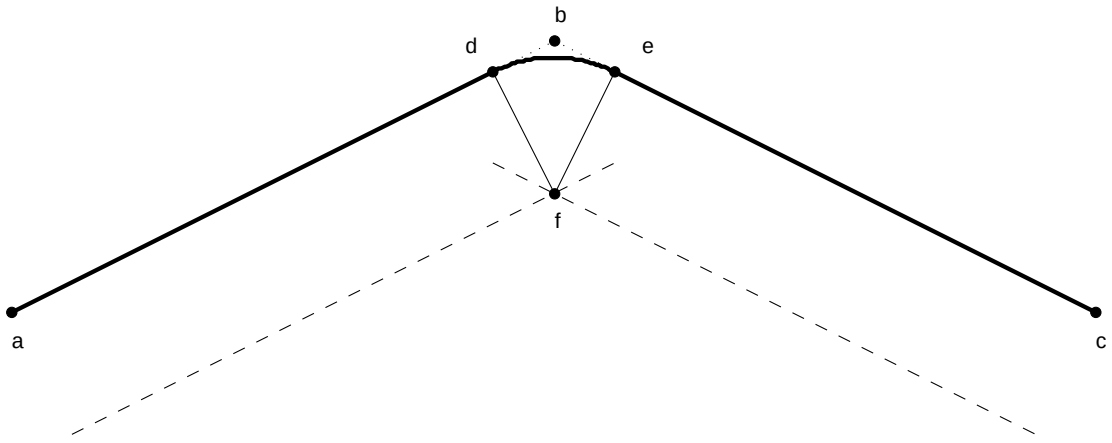

Figure 3.4. Filleted arc, obtuse waypoint path angle.

If $M = 0$, this is a degenerate case, resulting in

$$f = b. \tag{3.11}$$

Otherwise, the resulting arc center point is calculated by

$$f_x = \frac{\left|\begin{bmatrix} L_1 & (p_1 - p_2)_x \\ L_2 & (p_3 - p_4)_x \end{bmatrix}\right|}{M}, \tag{3.12}$$

and

$$f_y = \frac{\left|\begin{bmatrix} L_1 & (p_1 - p_2)_y \\ L_2 & (p_3 - p_4)_y \end{bmatrix}\right|}{M}. \tag{3.13}$$

Where

$$f = \begin{bmatrix} f_x \\ f_y \end{bmatrix}. \tag{3.14}$$

The endpoints of the filleted arc, d and e, as shown in Figure 3.4 and Figure 3.5, are constructed by,

$$d = f - r \cdot \|\vec{df}\|, \tag{3.15}$$

$$e = f - r \cdot \|\vec{ef}\|. \tag{3.16}$$

Where r is the arc fillet radius, and $\|\vec{df}\|$ and $\|\vec{ef}\|$ are defined in (3.1) and (3.2) respectively.

The angle $\angle def$, γ is calculated by the following:

$$\gamma = \frac{acos((d - f)\dot{(}e - f))}{\|(d - f)\| \cdot \|(e - f)\|}. \tag{3.17}$$

The direction of the curve is needed for the closed form solution.

The length of the shortened part of the two original line segments is given by,

$$l = \|(b - d)\| = \|(b - e)\|. \tag{3.18}$$

3.3 Lines and Circular Arcs

The baseline path generation method is the use of constant radius arcs. These paths are C^0 curvature continuous. The discontinuity in the change of curvature between line and arc segments requires that a robot must stop at each curve boundary point to satisfy kinematic constraints. In practice, a robust controller can allow a robot to track a C^0 continuous path with bounded error.

Constant radius circular arcs are desirable because of their geometric properties, such as endpoint tangency and the existence of straightforward closed form solutions for path parameterization.

The arc radius is chosen in consideration of trajectory velocity requirements. As velocity is higher, arc radius must also be higher due to centripetal acceleration, given by,

$$a_c = -\frac{v^2}{r}. \tag{3.19}$$

Where v is the linear velocity of the robot, and r is the radius of the circular arc. The maximum allowable lateral acceleration, a_{max} is related to the wheel ground contact friction force of the robot,

$$a_{max} = \mu \cdot g. \tag{3.20}$$

Where μ is the friction coefficient, and g is acceleration due to gravity. Substituting (3.20) into (3.19) yields

$$r = \frac{v^2}{\mu \cdot g}. \tag{3.21}$$

For example, given a friction coefficient $\mu = 0.4$, and velocity $v = 1.0\text{m/s}$,

$$r = \frac{(1.0\text{m/s})^2}{(0.4) \cdot (9.8\text{m}/s^2)}, \tag{3.22}$$

$$r = 0.26\text{m}. \tag{3.23}$$

In accordance with these constraints, an arc fillet radius of 0.25 meters is chosen for most trajectories. Velocity can likewise be constrained by arc radius, especially with curves such as Cornu spirals and splines. Solving (3.21) for v yields,

$$v = \sqrt{r \cdot \mu \cdot g}. \tag{3.24}$$

3.4 Polynomial Spirals

Polynomial spiral arcs are chosen as replacements to constant radius circular arcs for greater curvature continuity. These curves are represented by curvature as a function of arc length,

$$\kappa(s) = a_0 \cdot s^0 + a_1 \cdot s^1 + \ldots a_n \cdot s^n, \tag{3.25}$$

Where κ is the curvature in meters^{-1}, a_i is from a list of coefficients, and n is the order of the curve.

An arbitrary order polynomial is constructed, and its coefficients are solved to meet boundary conditions given by the filleting of two intersecting line segments. The two endpoints of the curve must coincide with the endpoints of the adjoining trimmed line segments, and the curvature at each endpoint must be zero.

The lack of a closed form solution increases the computational complexity.

A first order polynomial spiral is called a Clothoid, as discussed in Section. An example of a Clothoid is given in Figure .

To create a curve in Cartesian space, the curvature from polynomial equation describing the spiral must be used, along with a velocity profile, to solve the robot kinematics, as discussed in Section 4.2.

3.5 Quintic Splines

Quintic splines may improve curvature continuity. Splines can take the place of constant radius circular arcs or polynomial spirals to build curve segments that have specific boundary conditions of position, velocity, acceleration, and curvature. The manipulation of control polygons when generating these curves allows for these parameters to be controlled. The design, specification, and parameterization of quintic splines is discussed in this section.

An example of a filleted arc replaced with a quintic spline is given in Figure 3.6. The associated curvature profile is shown in Figure 3.7. The first and second derivatives of this curve are continuous, resulting in C^2 curvature continuity. A constant radius circular arc is shown for comparison. The change in curvature is minimized, but the maximum curvature is increased.

Figure 3.8 and Figure 3.9 show the first and second derivatives of curvature, respectively. The second derivative of curvature for a quintic spline is continuous, therefor guaranteeing C^2 continuity in curvature. This minimizes the wheel acceleration required for a robot to track this type of trajectory, improving performance over the line-arc trajectories discussed in Section 3.3.

Given arc parameters discussed in Section 3.2, a quintic spline is created. First, a control polygon is created, as shown in For a quintic spline, the control polygon has six points. The first and sixth points are the endpoints of the arc. The second point and third points control the first and second curvature derivatives of the arc. Both endpoints of every arc adjoin a straight line segment, with zero curvature and curvature derivatives. To ensure zero curvature and curvature derivatives at the boundaries of the arc, the first

and second, and fourth and fifth segments of the control polygon must be colinear with each other.

3.5.1 Parameterization

Splines are parameterized for trajectories using a subdivision algorithm, such as DeCastlejau's. With uniform sampling in parametric time, data point spacing in the trajectory parameteric data varies significantly. This causes reference trajectories to have too many data points in some areas, and too few in others. An iterative method may be used to parameterize a curve with uniform spacing in real time, as opposed to parametric time, as obtaining a closed form solution is not a trivial task.

A C^2 continuous spline path of chained quintic Bezier curve segments is defined by giving control points. Each Bezier segment is a 2D parametric curve, as given by

$$C(t) = [x(t), y(t)]\,, \tag{3.26}$$

with $t \in [0, 1]$ as a polynomial of $P(i)$ where $i = 1 \ldots 6$.

Evaluation of position and derivatives at a given position along the curve is achieved by using De Casteljau's algorithm. This algorithm evaluates points in the interior of a Bezier curve by subdivision.

The first and second derivatives of a quintic Bezier curve at the start point are,

$$C'(0) = 5(P_2 - P_1), \tag{3.27}$$

and

$$C''(0) = 20((P_1 - P_2) + (P_3 - P_2)). \tag{3.28}$$

The symmetric construction is used at the end point.

The desired linear velocity $v(t)$ of the robot is a function of the curvature at a specific point, described below. The parametric step in t along the curve that is necessary to produce that given geometric time step f^{-1} is initially estimated from the length of the initial tangent vector:

$$dt = f \cdot |C'(0)|/v(t). \tag{3.29}$$

The chord length distance to the next point along the curve is computed by

$$d = |C(t + dt) - C(t)|. \tag{3.30}$$

The parametric speed along the curve changes independently from the desired velocity. To solve this problem, dt is adaptively refined to match d to the desired step velocity

within a close tolerance of $d \approx v(t)/f$. A bisection refinement algorithm is used, as described in This refinement is repeated for each point along the curve, typically needing only one or two iterations because the velocity of the curve changes smoothly.

The following Cartesian states $x(t)$, $y(t)$ and $\phi(t)$ are calculated for each time t. To calculate ϕ, the nonholonomic constraint, (4.2), is applied. A tangential velocity is defined for the boundaries of each curve. The desired velocity at each time increment will be interpolated from these values. Two additional parameters are needed for trajectory specification, $v(t)$ and $\omega(t)$, which are the linear and angular velocities.

Curvature at each time t is defined by

$$\kappa(t) = (x'y'' - y'x'')/((x'^2 + y'^2)^{3/2})..\tag{3.31}$$

Velocity along the curve is computed from curvature through

$$v(t) = \frac{v_{max}}{1 + \frac{\kappa(t)^2}{2}},\tag{3.32}$$

where v_{max} is defined as the maximum desired velocity. The rotational velocity ω is calculated through velocity and curvature by

$$\omega(t) = v(t) \cdot \kappa(t).\tag{3.33}$$

The trajectories built by components in this chapter are sent to a controller, which then creates commands to be sent to the robots. In the following chapter, various motion controllers are designed and discussed.

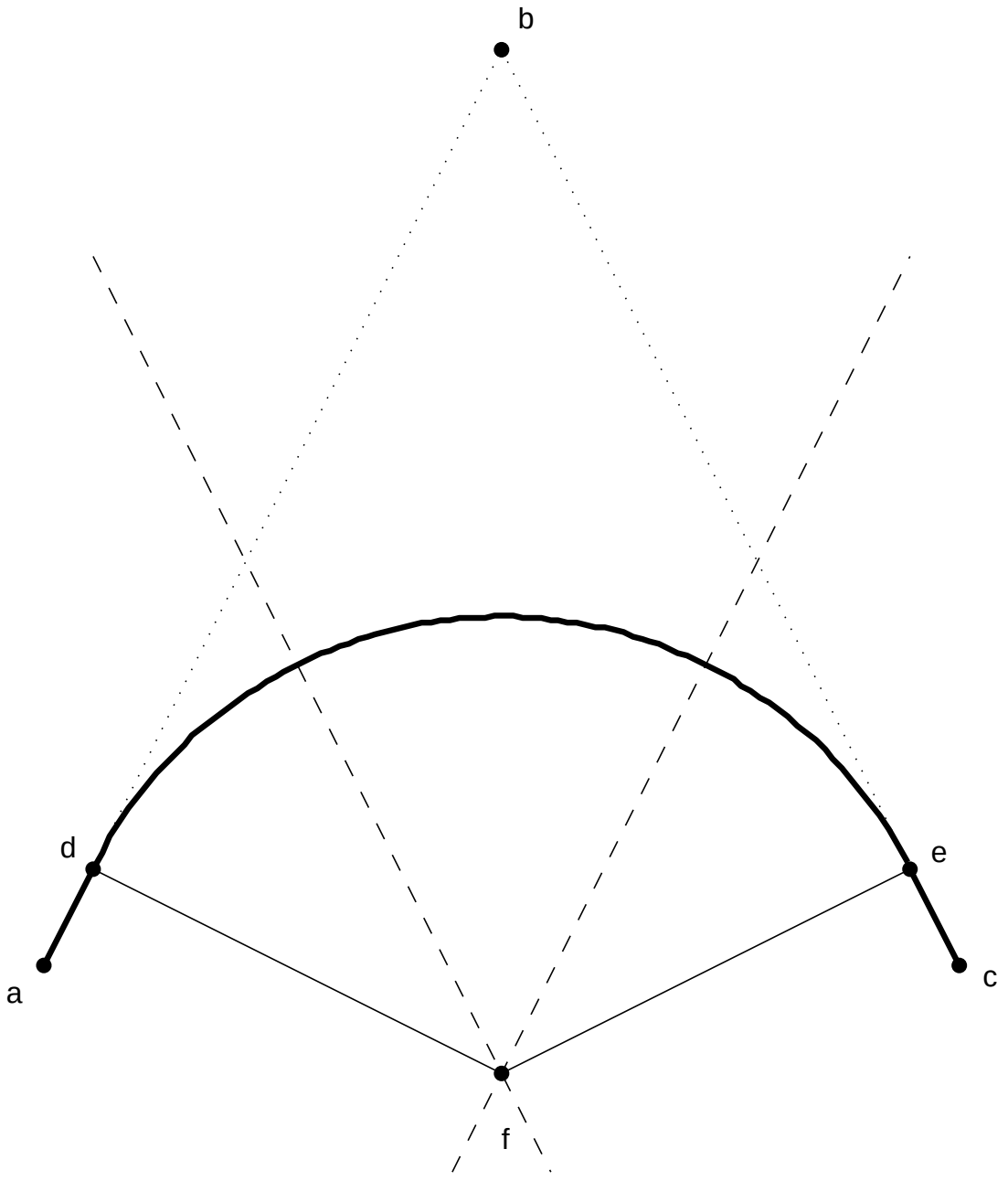

Figure 3.5. Filleted arc, acute waypoint path angle.

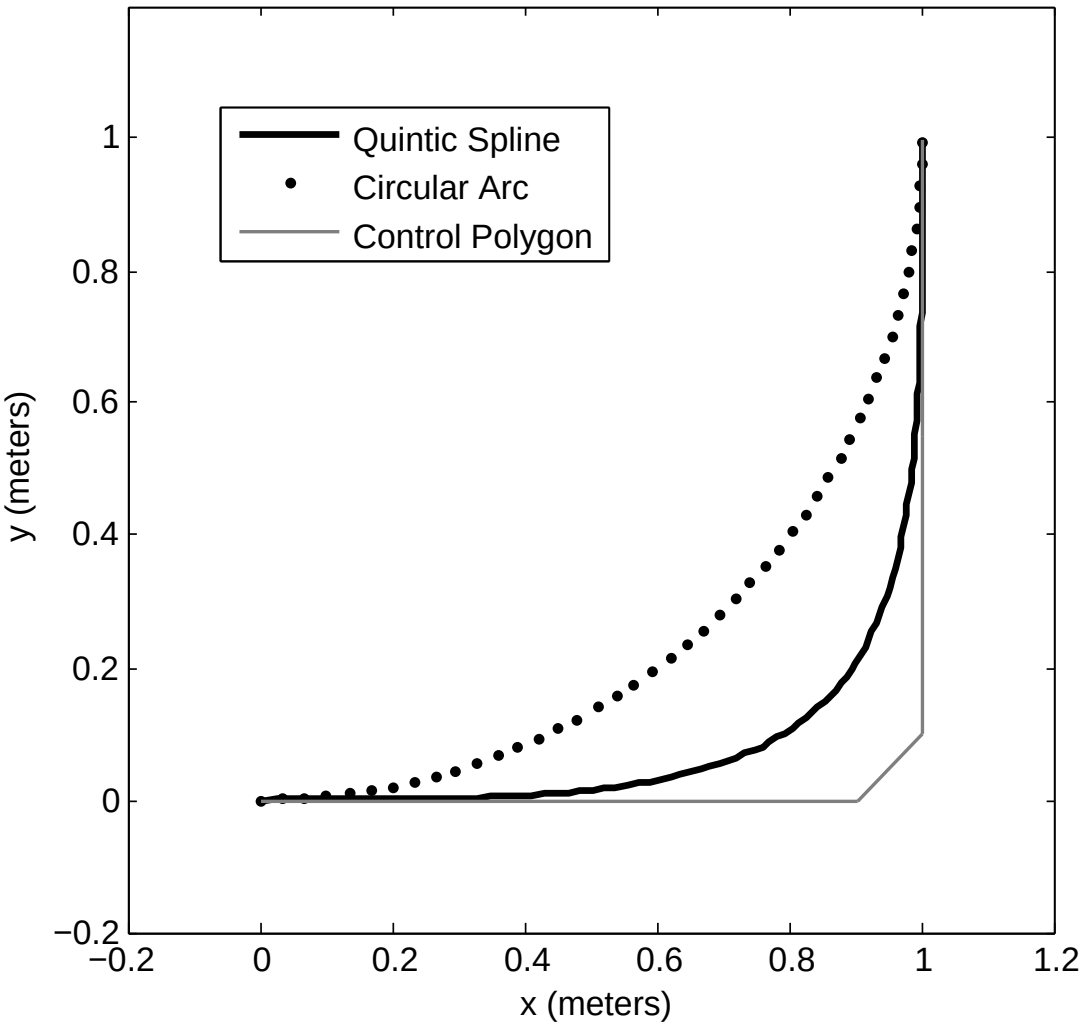

Figure 3.6. Example of a quintic spline.

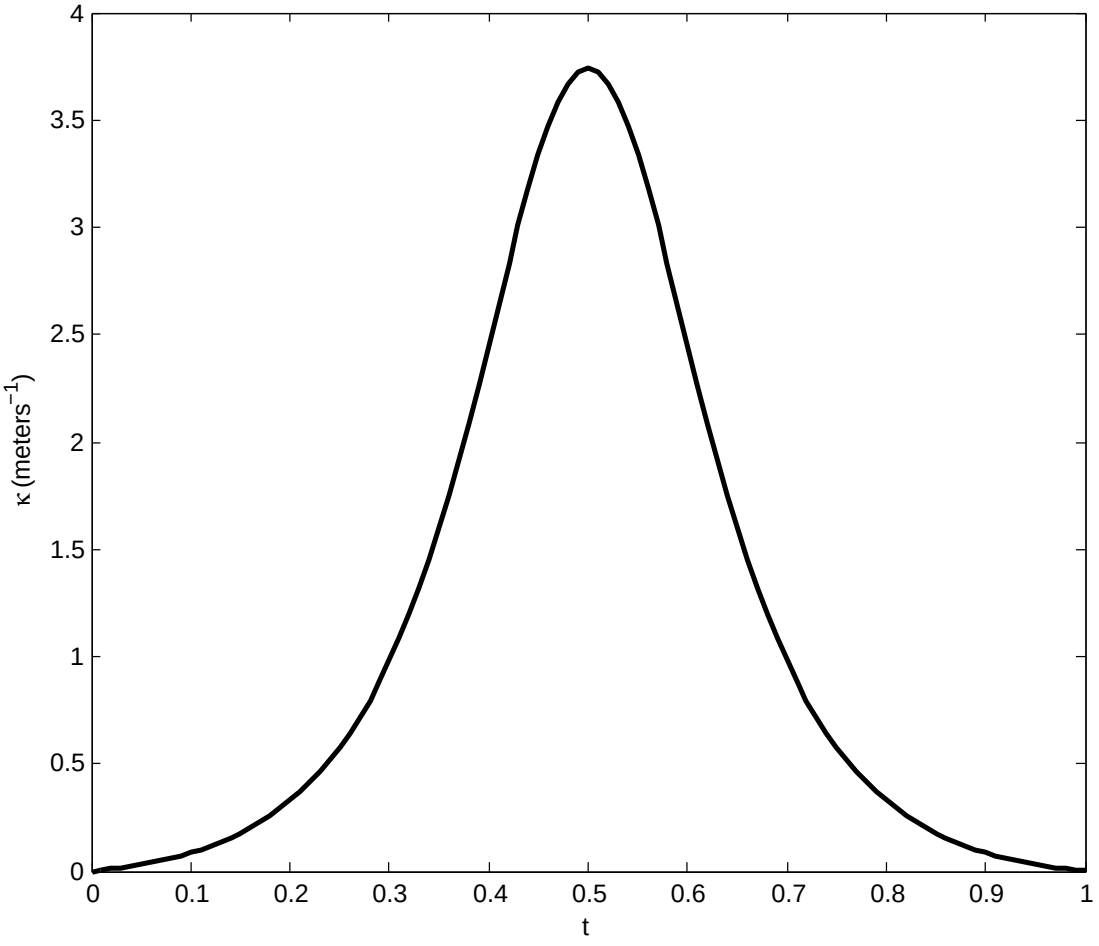

Figure 3.7. Curvature profile of spline shown in Figure 3.6

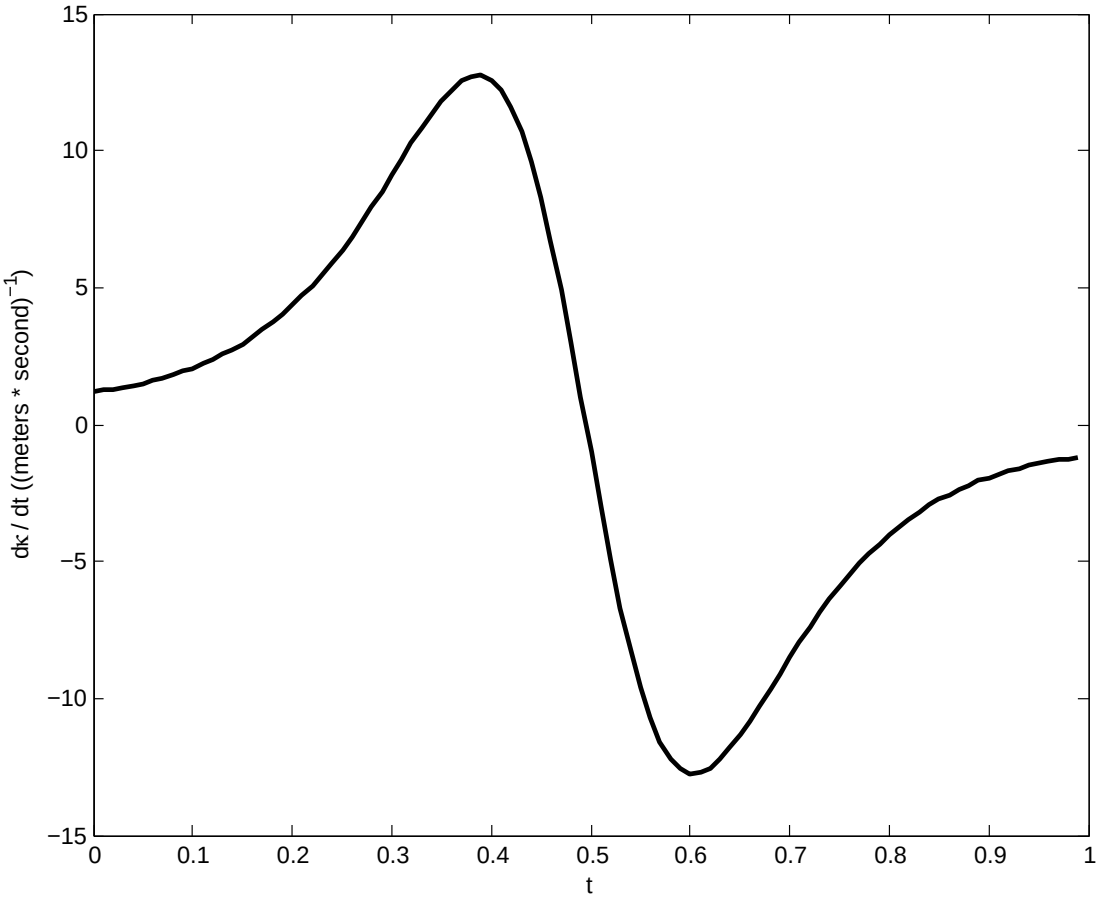

Figure 3.8. Derivative of curvature of spline shown in Figure 3.6

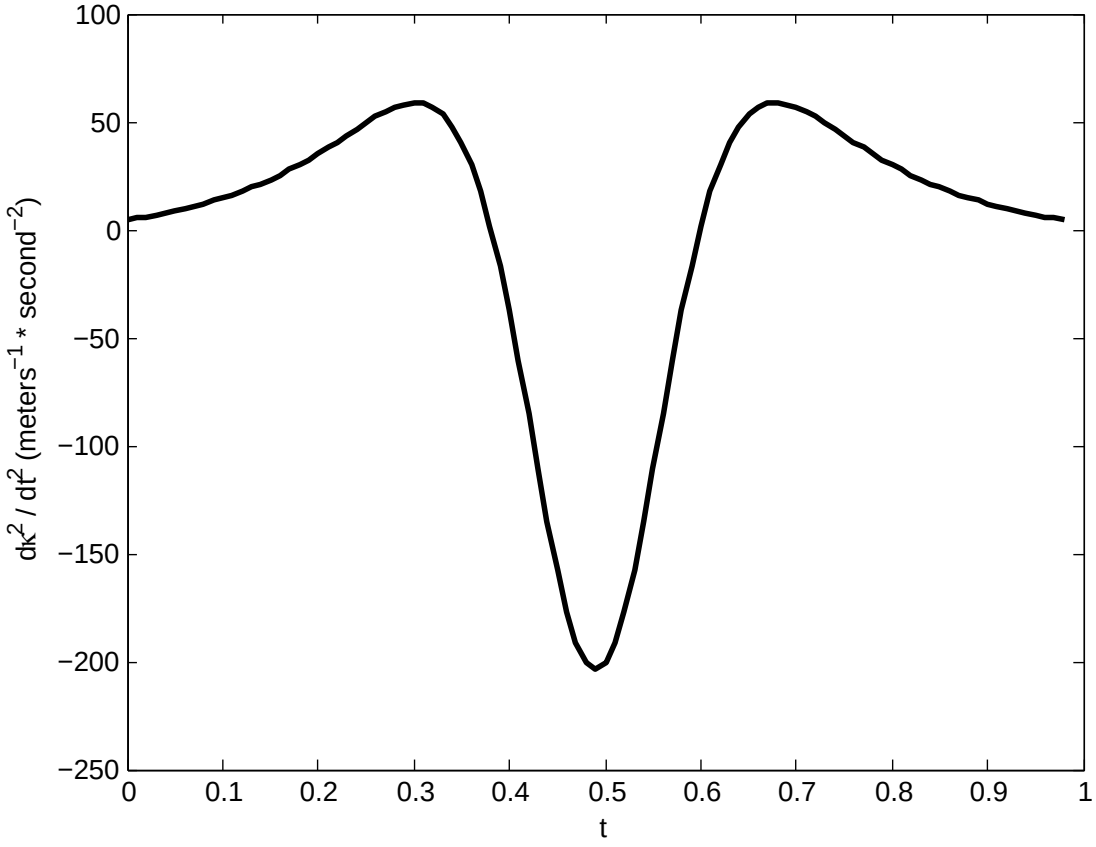

Figure 3.9. Second derivative of curvature of spline shown in Figure 3.6

CHAPTER 4

MOTION CONTROL

Given a path plan, such as a goal point or trajectory, a controller is required to drive a robot towards its objective. The availability of computing hardware for control of the robots in Emulab Mobile allows for motion control schemes to be tested and implemented fully in software. Accurate localization of robots enables the use of state feedback control. These attributes maximize the design flexibility of the system, but also are partially responsible for the constaints discussed in Section 1.3. The use of software control in this instance detaches the motion controller from the robot hardware, increasing the sensitivity of the system to imperfect communication channels.

The initial implementation of robot motion control in Emulab Mobile, presented in Section 4.1, made use of the vendor provided application programming interface for the Garcia robots. Motion is restricted to straight line segments and zero radius pivots. This works well for line segment paths with no arcs.

A posture stabilizing controller is presented in Section 4.3. This controller regulates a robot to a single goal point, and uses continual state feedback from the localization system discussed in Subsection 2.5.3. The implementation of this controller was used to test feedback control on the testbed, and establish that a trajectory tracking controller would ultimately be feasible on the system.

To increase velocities, and decrease robot time in motion, more advanced reference trajectories are employed, as discussed in Chapter 3. The robot kinematics, as associated with these trajectories is discussed in Section 4.2. A state feedback trajectory tracking controller is introduced in Section 4.4, which supersedes the *primitive* motion model as the default motion controller for Emulab Mobile.

4.1 Primitive Motion

The initial effort into robot motion on Emulab Mobile is point to point motion, accomplished using vendor supplied motion commands. These commands, termed prim-

itives provide the basic elements needed to move in straight lines, pivot, and turn. The primitives take as input length and angle measurements, and execute low level motion commands using odometry as a reference.

The robot application programming interface has methods to allow for the configuration of parameters related to motion primitives. For example, there are settings for maximum wheel velocity, wheel stall threshold, and wheel acceleration. These parameters are tuned depending on operating conditions and performance requirements.

When motion primitives are executed, termination is triggered by set boundary conditions. For a *move* primitive, which moves a robot a set distance in a straight line, the motion is terminated under the following conditions:

- Odometry idicates that the robot has traveled by the set distance
- A wheel has stalled
- A proximity sensor is triggered
- A cliff sensor is triggered
- The primitive is aborted

The relevent motion primitives in this research are *move* and *pivot*. With these two motions, a robot can be sent to any single goal posture in Cartesian space. Refer to Section 7.1 for more information on the implementation of robot coordination using primitives.

To achieve faster and more accurate goal posture attainment, state feedback control is used. This relies on the *null* primitive, which allows instantaneous wheel velocities to be commanded. Kinematic control may be implemented directly. The following sections in this chapter discuss the design of kinematic controllers to accomplish posture stabilization, path following, and trajectory tracking, all utilizing the *null* primitive.

4.2 Robot Kinematics

The robots used for Emulab Mobile are exclusively differentially steered wheeled mobile robots. Each robot has two independently controlled large wheels, on sides opposite along the longitudinal axis of the robot, sharing the same transverse axis. A passive two degree of freedom roller wheel is used to statically support the rear of the robot. A unicycle kinematic model is used for this class of robot. This model may be visualized as a rolling disk on a flat plane.

4.2.1 Cartesian Kinematic System

Three robot position states, x, y, and ϕ are used to denote Cartesian position and orientation within a known reference frame. In the case of posture regulation, the origin of the coordinate system coincides with the goal posture. Otherwise, the origin is arbitrarily chosen as a known datum point in the workspace. The robot state equations in Cartesian space are

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v \cdot cos(\phi) \\ v \cdot sin(\phi) \\ \omega \end{bmatrix}. \tag{4.1}$$

The system inputs, v and ω denote linear and rotational velocities respectively. There are three system states, but only two inputs. This class of wheeled mobile robot has a nonholonomic kinematic constraint given by,

$$x \cdot sin(\phi) + y \cdot cos(\phi) = 0. \tag{4.2}$$

This describes the constraint that the robot may only move linearly along its longitudinal axis, (input v), or in rotation, (input ω). An arbitrary set of states, $[x_1 y_1 \phi_1]$ may not necessarily be reached from another initial arbitrary set of states, $[x_0 y_0 \phi_2]$ by a single straight line motion.

4.2.2 Polar Kinematic System

To overcome some of the drawbacks associated with the Cartesian system representation, a Polar representation is used. A Cartesian to Polar state transformation is given by

$$\begin{bmatrix} e \\ \theta \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ atan2(-y, -x) \\ \theta - \phi \end{bmatrix}, \tag{4.3}$$

where x, y, and ϕ are the Cartesian states, and e, θ, and α are the Polar states. A kinematic diagram of the Polar states e, θ, and α, Cartesian states x, y, and ϕ, and system velocity inputs v and ω is given in Figure 4.1. The axes **O** signify the goal reference frame, to which the wheeled robot is stabilized. For the posture stabilization problem, the polar state equations are

$$\begin{bmatrix} \dot{e} \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} v \cdot cos(\alpha) \\ \omega + v \cdot \frac{sin(\alpha)}{e} \\ v \cdot \frac{sin(\alpha)}{e} \end{bmatrix}. \tag{4.4}$$

For the trajectory tracking problem, three states for a reference trajectory in Cartesian coordinates are added to the system. These state equations are given by,

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\phi}_r \end{bmatrix} = \begin{bmatrix} v_r \cdot cos(\phi_r) \\ v_r \cdot sin(\phi_r) \\ \omega_r \end{bmatrix}, \tag{4.5}$$

which is in the same form as (4.1). With the addition of these reference posture states, the Polar states given in (4.3) become,

$$\begin{bmatrix} e \\ \theta \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{(x - x_r)^2 + (y - y_r)^2} \\ atan2(-(y - y_r), -(x - x_r)) - \phi_r \\ \theta - \phi + \phi_r \end{bmatrix}. \tag{4.6}$$

Figure 4.2 illustrates the kinematics of the trajectory tracking problem. The reference frame is indicated by a grey set of wheels, while the actual robot is indicated by the black set. The reference trajectory is given by the dashed line, and the actual trajectory is a solid black line. The Polar states e, θ, and α are given, along with their Cartesian components and system inputs (4.6).

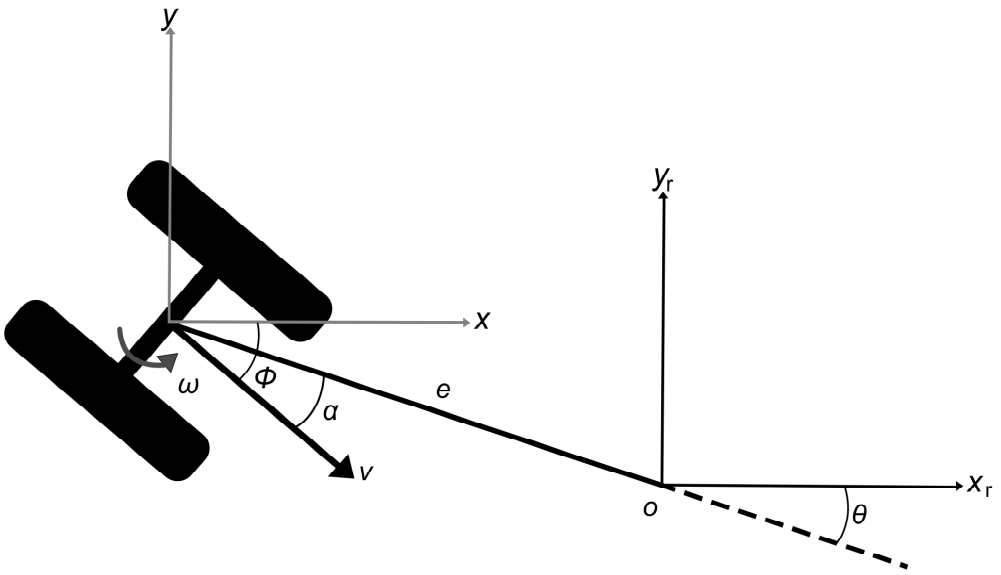

Figure 4.1. Polar kinematic diagram for posture stabilization.

As presented in [14], polar state equations are defined. Differentiating (4.6) and substituting (4.1), (4.5) and (4.6), (4.6) becomes

$$\begin{bmatrix} \dot{e} \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} v \cdot cos(\alpha) + v_r \cdot cos(\theta) \\ v \cdot \frac{sin(\alpha)}{e} - v_r \cdot \frac{sin(\alpha)}{e} - \dot{\phi}_r \\ v \cdot \frac{sin(\alpha)}{e} - v_r \cdot \frac{sin(\alpha)}{e} - \dot{\phi} \end{bmatrix}. \tag{4.7}$$

The two polar systems defined in (4.4) and (4.7) are used for the design and simulation of the posture stabilizing controller and trajectory tracking controller, respectively. In simulation, these differential equations are solved directly, while in implementation the Cartesian to Polar transformations are used to directly calculate the Polar states.

4.2.3 Kinematic Constraints

The Garcia robots used in Emulab Mobile have specific kinematic constraints related to their dimensions and configurations. While a differentially steered wheeled mobile robot is capable of a zero radius turn, (IE a pivot), there exist curvature constraints based on control and traction requirements. In this subsection, the dimensional constraints of the robots, and how they relate to the robot kinematics are discussed.

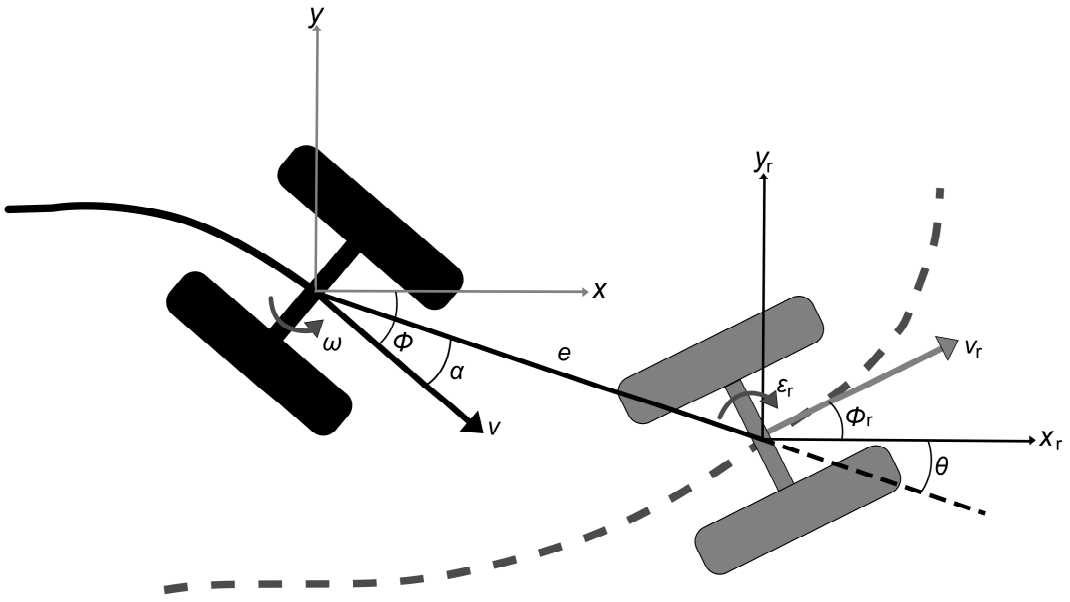

Figure 4.2. Polar kinematic diagram for trajectory tracking.

The nonlinear controllers discussed in this chapter, as kinematic motion controllers, produce commands in the form of velocities. Specifically, the control laws presented in this research produce the inputs v and ω for the systems described in the previous subsections. The low level control systems on the robot perform velocity tracking on a per wheel basis. This requires that wheel velocities be calculated for the v and ω velocities output from the motion controllers. The system inputs v and ω can be calculated from the individual wheel velocities by,

$$v = \frac{v_L + v_R}{2}, \tag{4.8}$$

$$\omega = \frac{v_R - v_L}{2 \cdot R}. \tag{4.9}$$

By solving (4.8) and (4.9) for v_L and v_R, the individual left and right wheel velocities are calculated by,

$$\begin{bmatrix} v_L \\ v_R \end{bmatrix} = \begin{bmatrix} v - \frac{R\omega}{2} \\ v + \frac{R\omega}{2} \end{bmatrix}, \tag{4.10}$$

where $R = 0.0889$ meters, as given by Table 2.1.

At maximum wheel velocity,

$$v_{max} = v + |\omega| \cdot R \tag{4.11}$$

Letting $v_{max} = 1.0$ meters per second, consider that $\omega_{max} = (v_{max} - v)/R$. This results in $\omega_{max} \approx 5.6243$ radians per second.

The Polar state θ as calculated by the Cartesian to Polar conversion in (4.3) is compared to θ calculated by the Polar state equation (4.4). Without correction for discontinuities in the Cartesian to Polar conversion, θ may be $\pm\pi$ from the actual state value. To prevent discontinuities, unwrapping must be performed, which accounts for the periodicity of the two dimensional arctangent trigonometric function used to calculate θ.

4.3 Posture Stabilizing Controller

As follows in this section is the development of a smooth, state feedback linearizing controller for the purpose of posture stabilization. Using this controller, a nonholonomic system is regulated to a single equilibrium point coincident with a goal posture. The development of this controller is undertaken as an intermediate step in the development of robust motion control for Emulab Mobile. The ultimate goal for the motion controller is to support full trajectory tracking. The establishment of posture stabilization through

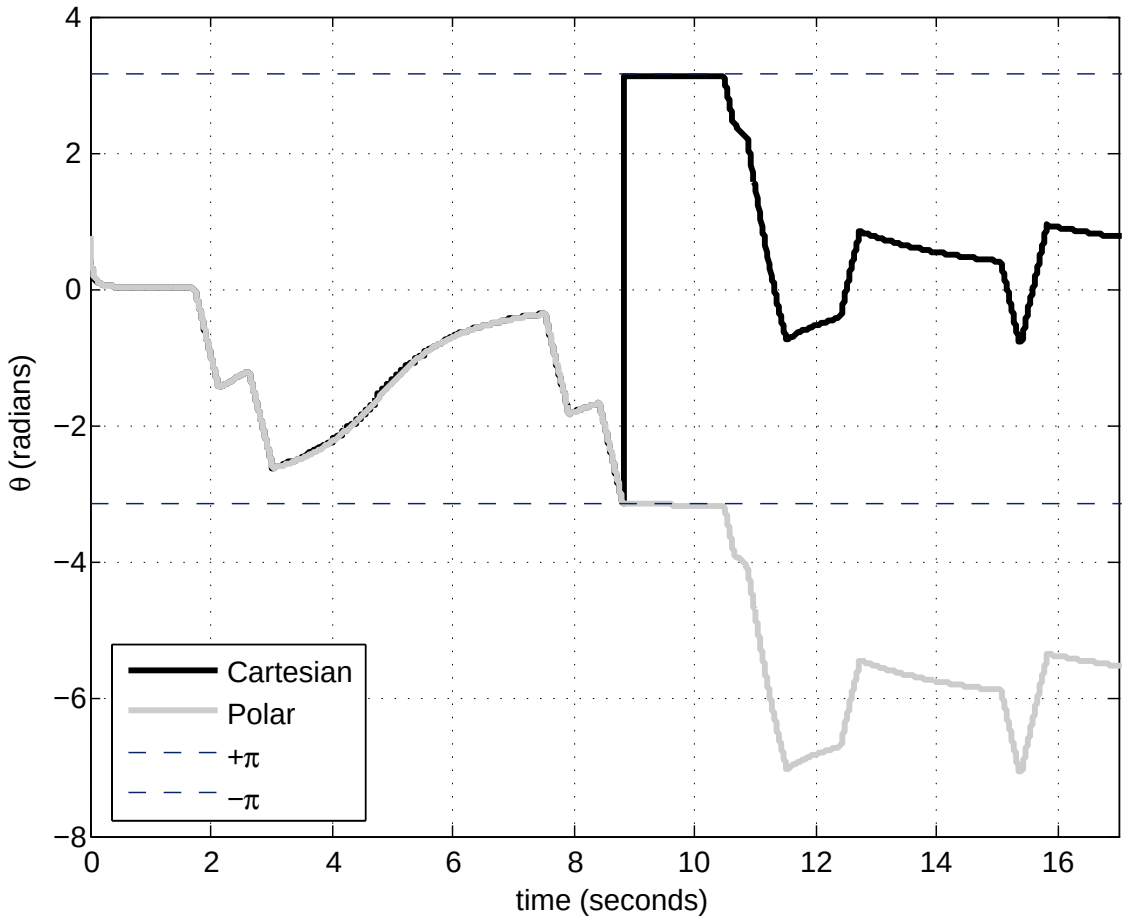

Figure 4.3. Comparison of θ calculation.

a nonlinear state feedback controller is used to test the feasibility of the use of this class of motion control on the testbed.

From the polar state equations (4.4), a control law for a state feedback controller,

$$[u, \omega]^T = g(e, \alpha, \theta), \tag{4.12}$$

is desired to drive states e, θ and α to zero. A control law is designed via Lyapunov analysis. A quadratic form Lyapunov candidate function,

$$V = V_1 + V_2 = \frac{1}{2}\lambda e^2 + \frac{1}{2}\left(\alpha^2 + h\theta^2\right), \tag{4.13}$$

is chosen. where e is the error distance vector, and $[\alpha, \sqrt{h} \cdot \theta^2]^T$ is the alignment error vector, and where $\lambda > 0$ and $h > 0$.

Taking the derivative of (4.13) results in,

$$\dot{V} = \dot{V_1} + \dot{V_2}, \tag{4.14}$$

which is split into two parts:

$$\dot{V_1} = \lambda e \cdot \dot{e}, \tag{4.15}$$

$$\dot{V_2} = \left(\alpha\dot{\alpha} + h\theta\dot{\theta}\right). \tag{4.16}$$

The control laws

$$v = (\gamma cos\ \alpha) \cdot e, \tag{4.17}$$

$$\omega = k\alpha + \gamma\frac{cos\ \alpha sin\ \alpha}{\alpha}\left(\alpha + h\theta\right), \tag{4.18}$$

are chosen, to be substituted into (4.19) and (4.23). The controller gains are constrained by $\gamma > 0$ and $k > 0$.

Considering $\dot{V_1}$ first, the state equations (4.3) and velocity control law (4.17) are substituted, resulting in,

$$\dot{V_1} = \lambda evcos(\alpha). \tag{4.19}$$

$$\dot{V_1} = \lambda ecos(\alpha)\left(\gamma cos(\alpha)\right)e, \tag{4.20}$$

$$\dot{V_1} = \lambda e^2 cos^2(\alpha)\gamma. \tag{4.21}$$

$$\dot{V_1} = -(\lambda sin^2(\alpha))e^2 \leq 0. \tag{4.22}$$

The state equations (4.3) and control laws (4.17), (4.18) are also substituted into (4.16).

Starting with,

$$\dot{V_2} = \alpha\left(-\omega + v\frac{sin(\alpha)}{e}\right) + h\theta\left(v\frac{sin(\alpha)}{e}\right), \tag{4.23}$$

results in,

$$\dot{V_2} = \alpha\left(-\omega + v\frac{sin(\alpha)}{\alpha}\frac{(\alpha + h\theta)}{e}\right), \tag{4.24}$$

$$\dot{V_2} = \alpha\left(-\omega + \left((\gamma cos(\alpha)e)\frac{sin(\alpha)}{\alpha}\frac{(\alpha + h\cdot\theta)}{e}\right)\right), \tag{4.25}$$

$$\dot{V_2} = \alpha\left(-\omega + \frac{\gamma cos(\alpha)sin(\alpha)}{\alpha}\left(\alpha + h\cdot\theta\right)\right). \tag{4.26}$$

$$\dot{V_2} = -k\alpha^2 \leq 0. \tag{4.27}$$

The derivate of the original Lyapunov function from (4.14) is assembled in (4.28). This is an abuse of the terminology, as the derivative is only negative semi-definite, instead of negative definite. The final result is,

$$\dot{V} = -\lambda \left(\gamma \cdot cos^2(\alpha)\right) e^2 - k\alpha \leq 0. \tag{4.28}$$

An alternate set of control laws are designed to permit forward motion only, eliminating cusps in the resulting path The avoidance of cusps is desirable in situations where the high torques required to quickly reverse, stop, and move forward may become a detriment to battery life. Avoidance of path cusps may also decrease the amount of time required for posture stabilization. Furthermore, it is aesthetically pleasing to testbed users when robots proceed along more direct paths to their destinations.

The alternate control laws,

$$v = v_{max} tanh\left(\frac{\gamma e}{v_{max}}\right), \tag{4.29}$$

and

$$\omega = \frac{(sin(\alpha))\left(1 + h\frac{\theta}{\alpha}\right) + \beta\alpha}{e}. \tag{4.30}$$

are developed. A hyperbolic tangent function is used to provide a smooth saturation of v_{max}.

4.4 Kinematic State Feedback Trajectory Tracker

The design of a control law to use for trajectory tracking is presented in Section 4.4.1. The design of a dynamic extension to this controller is discussed in Section 4.4.2. Parameter and gain specification of the control system is given in Section 7.4.2. The stability analysis of the discretized system is discussed in Section 6.3.4.

4.4.1 Control Law

A nonlinear control law to track a robot to a reference frame along a circular path manifold are developed using Lyapunov based techniques. This controller presented in section is capable of solving the posture stabilization, path following, and trajectory tracking problems simultaneously. The controller development and background, along with more details about the design of the circular path manifold are discused in [14].

Considering the system model given in Section 4.2.2, Quadratic Lyapunov candidate functions,

$$V_1 = \frac{1}{2}\left(e - r\sqrt{2}\sqrt{1 - cos(2\theta)}\right)^2, \tag{4.31}$$

and

$$V_2 = \frac{1}{2}(\theta + \alpha)^2 \tag{4.32}$$

are specified, where $r\sqrt{2}\sqrt{1 - cos(2\theta)}$ is the circular path manifold. This path manifold is modified to produce,

$$e = r\sqrt{2}\sqrt{\zeta - cos(2\theta)}, \tag{4.33}$$

where $\zeta = 1 + \epsilon$. The term ϵ is a sufficiently small perturbation. This path manifold design avoids a singularity in the control law which occurs at $\theta(0) = 0$, as discussed in [14].

Defining $z = e - r\sqrt{2}\sqrt{\zeta - cos(2\theta)}$, (4.31) becomes,

$$V_1 = \frac{1}{2}z^2. \tag{4.34}$$

Taking the derivatives of V_1 and V_2 results in,

$$\dot{V_1} = z\left(v_r cos(\theta) - v cos(\alpha) - \frac{r\sqrt{2}sin(2\theta)}{\sqrt{\zeta - cos(2\theta)}}\left(\frac{v sin(\alpha) - v_r sin(\theta)}{e} - v_r\kappa_r\right)\right), \tag{4.35}$$

$$\dot{V_2} = (\theta + \alpha)\left(2\dot{\theta} + \dot{\phi_r} - \dot{\phi}\right). \tag{4.36}$$

As discussed in [14], the control law is evaluated based upon,

$$\dot{V_1} = -k_1 z tanh(z) \leq 0, \tag{4.37}$$

$$\dot{V_2} = -k_2(\theta + \alpha)\, tanh(\theta + \alpha) \leq 0, \tag{4.38}$$

where k_1 and k_2 are the controller gains. For stability, k_1 and k_2 must both be greater than zero. If this condition is met, the derivatives of the Lyapunov candidate function are negative definite, which results in asymptotic stability.

The optimized control law for linear velocity [14] is given as,

$$v = \frac{k_1 \cdot e \cdot k_e \cdot tanh(e - r\sqrt{2} \cdot k_e) + v_r \cdot e \cdot cos(\theta) \cdot k_e + v_r \cdot k_r \cdot (sin(\theta) + \frac{\omega_r}{v_r} \cdot e)}{e \cdot k_e + k_r \cdot sin(\alpha)}. \tag{4.39}$$

Where k_e is defined as

$$k_e = \sqrt{\zeta - cos(2\theta)}, \tag{4.40}$$

$$k_r = r\sqrt{2} \cdot sin(2\theta). \tag{4.41}$$

The optimized control law governing rotational velocity is given by,

$$\omega = k_2 \cdot tanh(\theta + \alpha) + 2\dot{\theta} + \dot{\phi_r}. \tag{4.42}$$

4.4.2 Dynamic Extension

The dynamic extension is defined by new states,

$$\dot{v} = -k_v(v - v_r) + \dot{v_r}, \tag{4.43}$$

$$\dot{\omega} = -k_\omega(\omega - \omega_r) + \dot{\omega_r}. \tag{4.44}$$

introduced to decrease steady state error and improve boundedness. This extension to the system also acts as a low pass filter, which improves the controller response in the presence of noisy state feedback.

With the completion of the design of trajectory generation and motion control systems, one final component is required to create a complete robot coordination and control system for Emulab Mobile. An obstacle avoidance system is needed to coordinate the motion of multiple robots, and to avoid obstacles present in the robot workspace. The next chapter presents an obstacle avoidance system used for reactive motion planning. After the presentation of the obstacle avoidance system, the results of all robot motion components in simulation are presented.

CHAPTER 5

OBSTACLE AVOIDANCE

In the iterative goal point progression model, obstacle avoidance is accomplished through a modified visibility graph method, as discussed in Section 3.1. This method works well for point to point motion and posture stabilization, but a better model is needed for path following and trajectory tracking. A method of smooth and continuous path generation for mobile robots maneuvering in a planar environment with multiple obstacles is presented in this chapter. The method is based upon construction of nonlinear dynamic phase portraits where key mathematical features of their underlying differential equations are manipulated in order to provide a novel trajectory generator resolving a number of known issues in the literature. Termed the Virtualized Phase Portrait Method (VPPM), this algorithm uses a planar velocity field instead of a scalar potential field. This provides trajectories devoid of oscillations, eliminates problems with local minima, and results in more direct control over bounded and smooth velocity and path curvature.

In the VPPM, a single vector of equations is constructed to describe the goal point and obstacles. The goal point appears in the phase portrait differential equations as a globally asymptotically stable node that attracts the robot directly to its desired goal point along a minimum length straight line path. Obstacles appear in the equations in order to deflect the robot from this ideal path as the robot approaches the obstacles. Due to the smooth saturation nature of the goal point and obstacle proximity effects, the trajectory velocity and curvature is ultimately smooth and bounded. Obstacle proximity effects also allow multiple obstacles to be considered simultaneously by one set of equations. In cases where groups of obstacles could trap or hinder the robot path, a bounding volume hierarchy is employed to combine obstacles. This allows concave regions to be filled.

A trajectory generated by VPPM through a workspace with moderately placed obstacles is shown in Figure 5.1. This illustrates key components of the velocity field, such as the goal sink, obstacle exclusion zones, obstacle regions, and the initial position. The obstacles in this example are placed manually, but could represent the configuration space

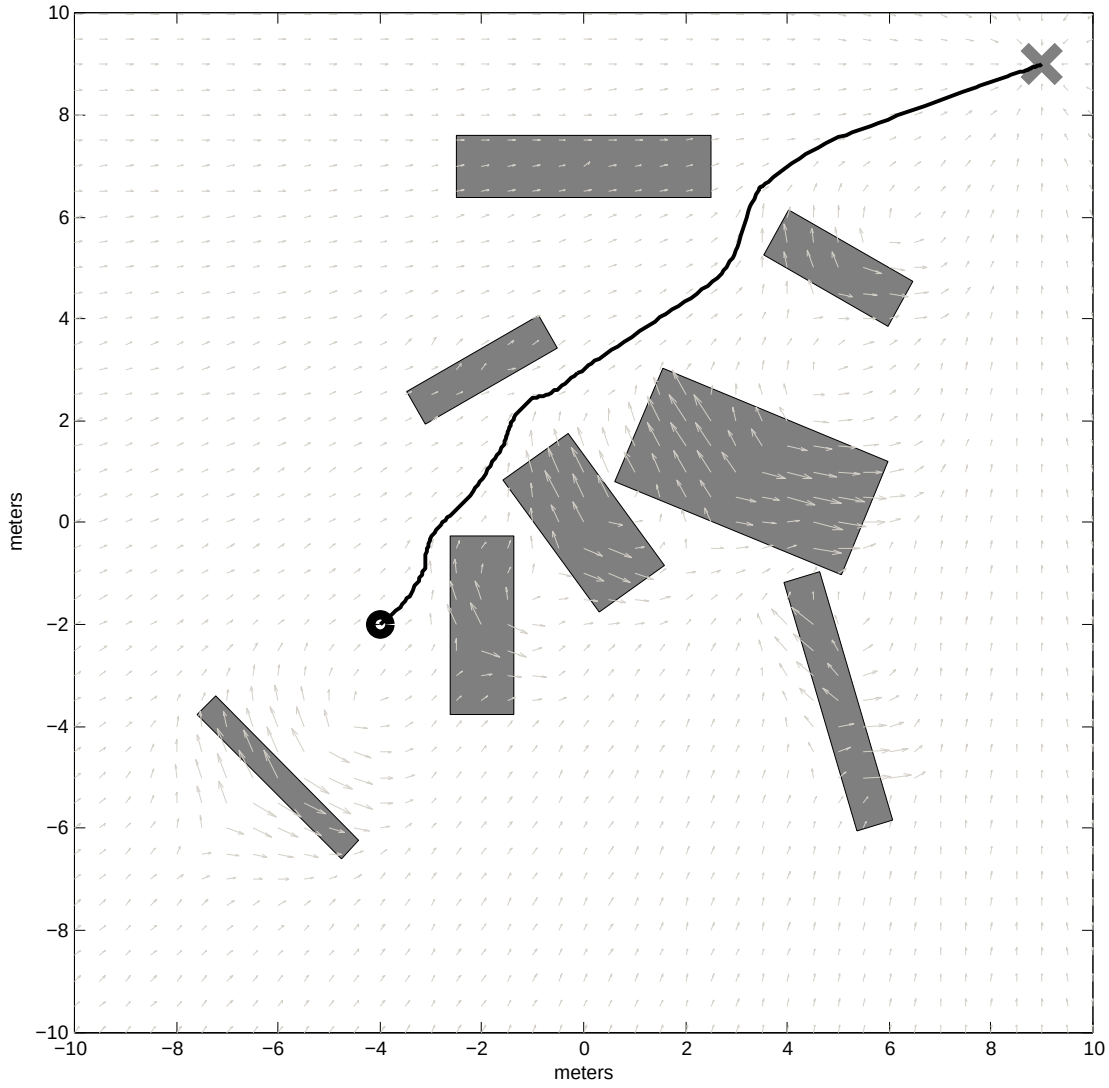

Figure 5.1. Robot trajectory simulated in a cluttered environment.

of a circular robot in an obstacle filled workspace known a priori.

5.1 Velocity Field Phase Portrait Method

Given the size and placement of obstacles and goal point location within the robot workspace, a virtual velocity field,

$$\begin{bmatrix} \dot{q}_x \\ \dot{q}_y \end{bmatrix} = \dot{q}_{goal} + \sum (\dot{q}_i), \tag{5.1}$$

is generated. The field is used to create a trajectory from an initial point to the final goal point. For simplicity within the scope of this article, the field equations are restricted to

$\Re^2$. The velocity field equation (5.1) shows the superposition of the goal attractor $\dot{q}_{goal}$, and obstacle repulsion fields $\dot{q}_i$. Where $i = 1, \ldots, n$, and n is the number of obstacles present.

5.1.1 Goal Sink

In this model, a single goal sink for each robot is considered. This goal sink globally attracts a robot to a specific Cartesian position within the workspace. A globally asymptotically stable equilibrium point is placed anywhere in $\Re^2$. The goal is asymptotically stable, not exponentially stable due to the usage of a saturation function to prevent undesirably strong velocity fields in areas distant from the sink. In proximity to the goal, the field strength weakens, by linearly decreasing the approach velocity of the robot near its objective to provide globally asymptotic convergence. As shown in Figure 5.2, all differential field lines are oriented towards the goal point.

The goal attractor function is a saturation function creating a field with all vectors oriented towards a single configured goal point. The goal attractor function could be substituted with a reference trajectory tracked by a nonlinear controller, or a goal posture regulator which drives a robot to a desired position and orientation.

A single Cartesian position, P_{goal} is configured for the goal attractor. The saturation level μ determines the magnitude of the velocity field converging to the goal point. A polar form saturation function orients the differential field vectors towards the goal point. The usage of two Cartesian saturation functions results in vectors converging to the major axes in outlying areas, with trajectories traveling along them to reach the equilibrium point at the goal. This behavior is undesirable in the interests of optimizing trajectory length.

The scalar polar distance magnitude,

$$e = \| \, q - P_{goal} \, \| \, , \tag{5.2}$$

and angle,

$$\theta = Atan2(q - P_{goal}), \tag{5.3}$$

are assembled to form the goal attractor field,

$$\dot{q}_{goal} = -\mu \cdot tanh(e) \cdot R_z(\theta) \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{5.4}$$

Where $R_z(\theta)$ is a standard two dimensional rotation matrix given by,

$$R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}. \tag{5.5}$$

Global asymptotic stability of the goal equilibrium point is thus guaranteed based upon Lyapunov techniques.

5.1.2 Obstacle Fields

Obstacle field overlays are created by defining a set of Cartesian center coordinates, Values describing the two dimensional size, orientation, and strength of the desired region of repulsion are also created. To define obstacle regions in the workspace, let $P_i \in \Re^2, i = 1, \ldots, n$ be the center positions, and $a_i \in \Re^2, i = 1, \ldots, n$ be the scalar exterior dimensions

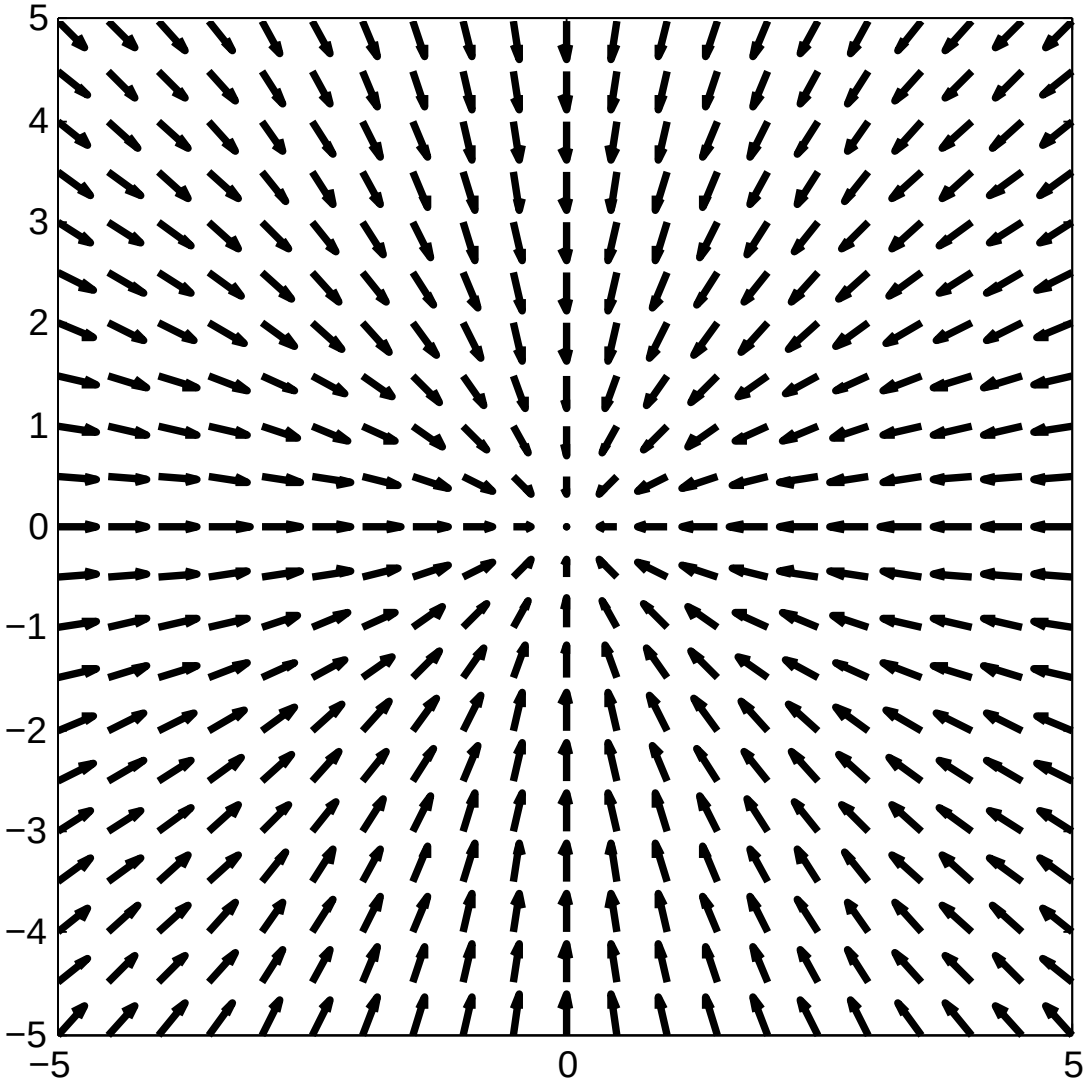

Figure 5.2. Field of a single goal sink at the origin.

of n rectangular obstacles. The orientation of each obstacle is given by ϕ.

A local coordinate system, $\hat{d}_i$, oriented by angle ϕ is created. The origin for each coordinate system is located at the center point of its parent obstacle.

The partial primary rolloff function,

$$\gamma_i = sat(g(\hat{y}/a_{max})), \tag{5.6}$$

establishes the shape and size of the obstacle repulsion field overlay. The term a_{max} is used for the maximum distance projected in $\hat{y}$, to normalize the primary rolloff function based on obstacle size and orientation. Examples of field functions are shown in Figure 5.3. The field is designed to inversely saturate in order to limit the influence of the obstacle field overlays to a controlled local area.

In Figure 5.3, the two solid line plots use the field roll off function $g(x) = x^2$, while the two dotted line trajectories use a field roll off function of $g(x) = x^4$. The black lines use the saturation function given by $sat(x) = 1 - tanh(x)$, and the gray lines use, $sat(x) = 1 - \left(\frac{2}{\pi}\right) \cdot atan(x)$. These functions are abbreviated as $tanh$ and $atan$ respectively, with the superscript referring to one of the above roll off functions by its exponent.

A saturation and roll off function pair is chosen to get the desired obstacle field overlay effect. For a strong, predictable obstacle region, it is best to use the $tanh^4$ field, (dotted black line). It has a sharp roll off, and diminishes at approximately 150% of the obstacle region distance from the center. For a smoother trajectory with lower curvature, it is best to choose the $atan^2$ field, (solid gray line). This field function has a large area of influence, which is desirable in regions with sparse obstacle disbursement. Other saturation and roll off function pairs can be chosen, as long as the region of local instability remains bounded.

The secondary roll off function acts in line with a trajectory approaching the goal position. It controls the region in which a trajectory is deflected before encountering an obstacle, as well as providing a smooth trajectory after the obstacle is cleared. A smooth field envelope is needed to prevent discontinuities in the velocity field. Minimizing the occurrences of discontinuities benefits the system by keeping the trajectory velocity bounded, and limiting curvature. Without a secondary rolloff function, the obstacle field acts in an unlimited region along the local x axis, as shown in Figure 5.4.

Originally, primary roll off functions acting along the adjoining major axis are used as secondary rolloff functions. With the addition of support for oriented obstacles, a new secondary rolloff function is needed. VPPM is extended with a new secondary rolloff

function, σ. This new extension not only provides support for oriented obstacles, but preserves continuity of the vector field, and improves interactions between closely spaced obstacles.

The secondary rolloff function, σ, is a scalar value calculated by solving a cubic polynomial parametric trajectory envelope around the obstacle. The current configuration q is transformed into obstacle local coordinates. For a given $\hat{x}$, a value in $\hat{y}$ is calculated. The magnitude of the trajectory envelope at that specific point is then used as the value for σ.

The obstacle local coordinate system rotation matrix is given by

$$R_\sigma = R_z(\phi - \theta + \pi). \tag{5.7}$$

The obstacle region is developed as an oriented bounding box, in a local coordinate system with an origin coincident to its center. The angle of orientation is normalized to lie between 0 and $\frac{\pi}{2}$. Any other orientations can be normalized to this range by switching the dimensions of the obstacle region. Using this rotation matrix, and the dimensional parameters for each individual obstacle, the vertices of the oriented bounding box are calculated,

$$V_{bottomleft} = R_\sigma * \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} * a_i/2, \tag{5.8}$$

$$V_{bottomright} = R_\sigma * \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * a_i/2, \tag{5.9}$$

$$V_{topleft} = R_\sigma * \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} * a_i/2, \tag{5.10}$$

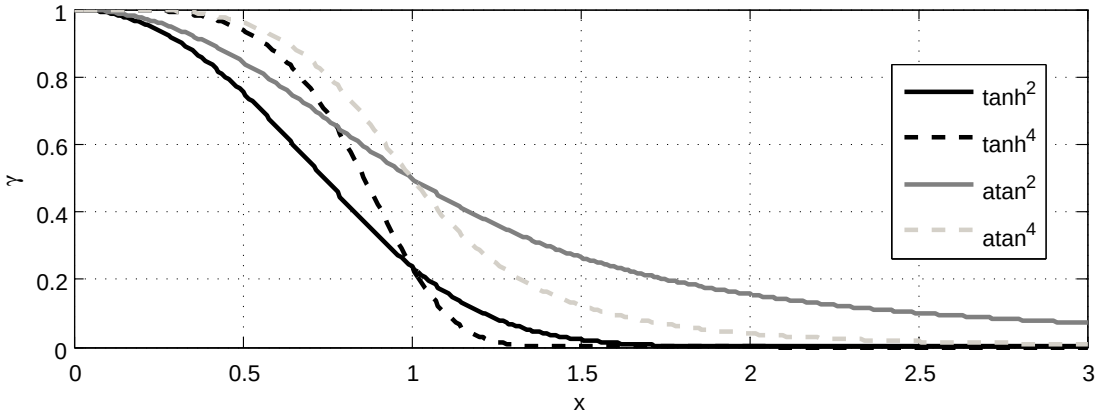

Figure 5.3. Rolloff of field functions γ_i as indicated.

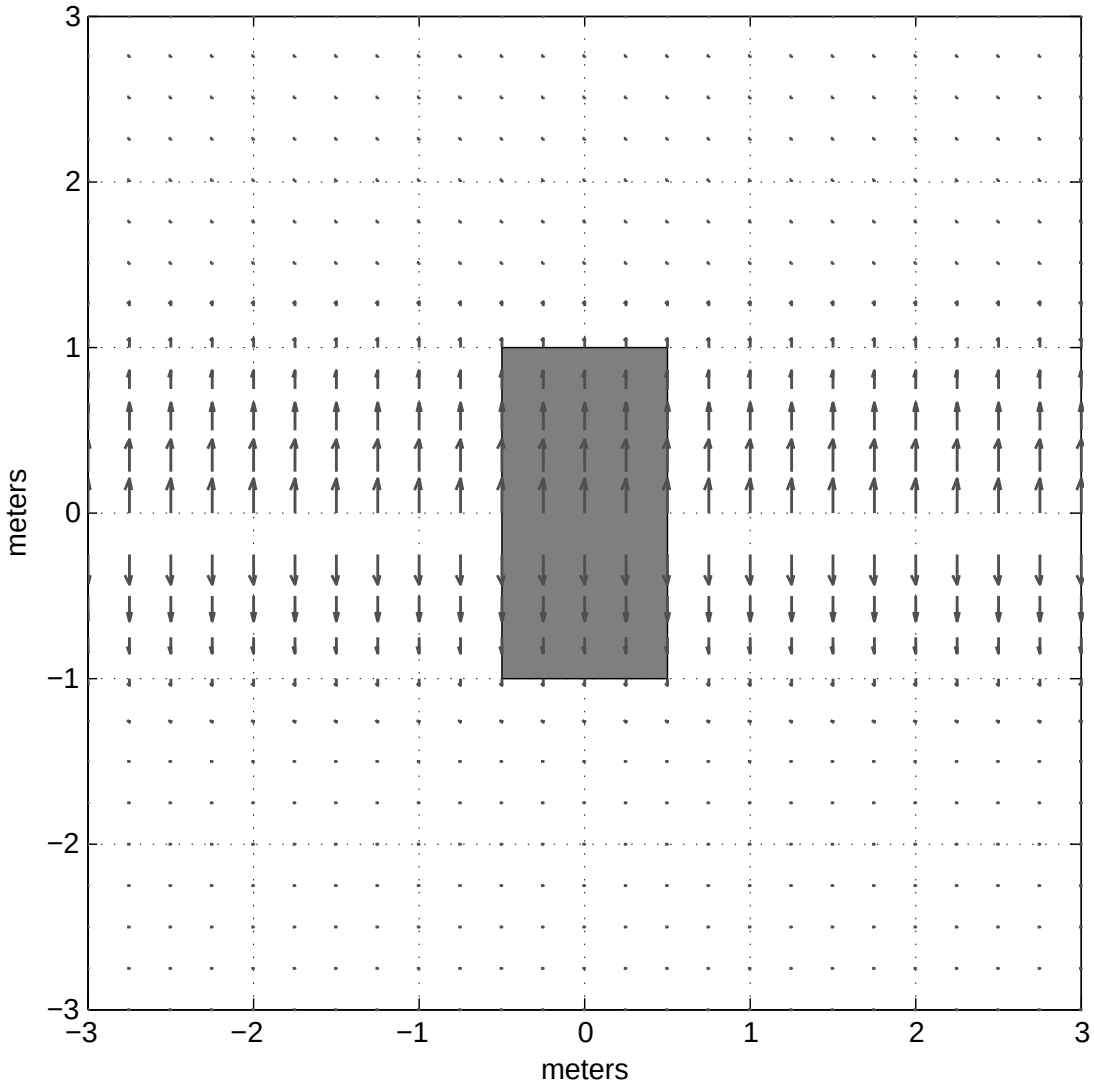

Figure 5.4. An obstacle with no secondary rolloff function.

$$V_{topright} = R_\sigma * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * a_i/2. \tag{5.11}$$

The term *bisection line* is used to denote the x axis of the local coordinate system. For any oriented bounding box given the above stated parameters, there can be only one or two vertex points above the bisection line. The bottom left vertex is always below the bisection line, and can be discarded from consideration for the secondary rolloff function. Likewise, the top right vertex is always above the bisection line, and is always considered in the secondary rolloff function. One of the remaining vertexes may lie above the bisection

line, and this is algorithmically determined in order to get the vertexes comprising the control points of the secondary rolloff function envelope.

The control points of an obstacle are determined by the one or two vertices above the bisection line, plus two more points along the bisection line to control the size of the obstacle approach and departure regions. These outlying control points are determined based on the velocity of the approaching trajectory, which in turn is influenced by the goal and local obstacle field strengths. Two scalar distance values, $w_{approach}$ and $w_{departure}$ are calculated, and used to determine the secondary rolloff envelope function control points.

The first control point is

$$G_1 = \begin{bmatrix} -w_{approach} \\ 0 \end{bmatrix}, \tag{5.12}$$

in the local obstacle coordinate system. If the top left obstacle vertex is above the bisection line, the next control point becomes,

$$G_i = V_{topleft}. \tag{5.13}$$

The top right vertex is always above the bisection line, which leads to its inclusion as a control point:

$$G_i = V_{topright}. \tag{5.14}$$

If the bottom right vertex is above the bisection line,

$$G_i = V_{bottomright}. \tag{5.15}$$

Due to the symmetric properties of oriented bounding boxes, either the top left or bottom right vertex can exclusively lie above the bisection line. If the orientation angle is $\frac{\pi}{4}$, both vertexes will lie directly on the bisection line, and will not be control points. The bottom left vertex is always below the bisection line, and therefor never considered as a control point. The final control point is given by,

$$G_i = \begin{bmatrix} w_{departure} \\ 0 \end{bmatrix}. \tag{5.16}$$

For each segment of the trajectory envelope, four parameters are given. The vectors P_i and P_f represent the initial and final control points, while v_i and v_f denote the derivatives which control the orientation at each control point. t is calculated given $\hat{x}$, which also

determines the active segment. For each segment, t starts at zero, and ends at one. The cubic trajectory coefficients for each segment are given by

$$k_1 = P_i, \tag{5.17}$$

$$k_2 = v_i, \tag{5.18}$$

$$k_3 = 3 * (P_f - P_i) - (2 * v_i + v_f), \tag{5.19}$$

and

$$k_4 = -2 * (P_f - P_i) + (v_i + v_f). \tag{5.20}$$

Using these coefficients yields

$$\hat{y} = k_4 \cdot t^3 + k_3 \cdot t^2 + k_2 \cdot t + k_1. \tag{5.21}$$

An example secondary rolloff function envelope for an obstacle oriented at zero is given in Figure 5.5. Four control points are used, and $w_{approach}$ and $w_{departure}$ are both arbitrarily set to one. Note that the top curve is the calculated envelope, and that the bottom curve is reverse-symmetric. In Figure 5.6, the same values are used, except that the obstacle is now oriented $\pi/4$. In this case, we only need three control points. The reverse symmetry properties are more prominent in this example.

The value for $w_{departure}$ is significantly decreased, and the orientation is changed to $\pi/6$ in Figure 5.7. A smaller exclusion zone around the obstacle is required in regions where the obstacle face slants away from the bisection line.

The partial obstacle field equation,

$$\rho = |\mu \cdot \sigma \cdot \gamma|, \tag{5.22}$$

is constructed from the primary and secondary rolloff functions, multiplied by the obstacle field strength parameter. Field lines are aligned perpendicular to the goal attractor to keep obstacle repulsion fields from influencing the approach velocity of a robot in proximity to an obstacle, which helps prevent local minima from forming. Otherwise, field lines radiating directly from the center of an obstacle region can create local minima on the side furthest from the goal attractor, while accelerating trajectories towards the

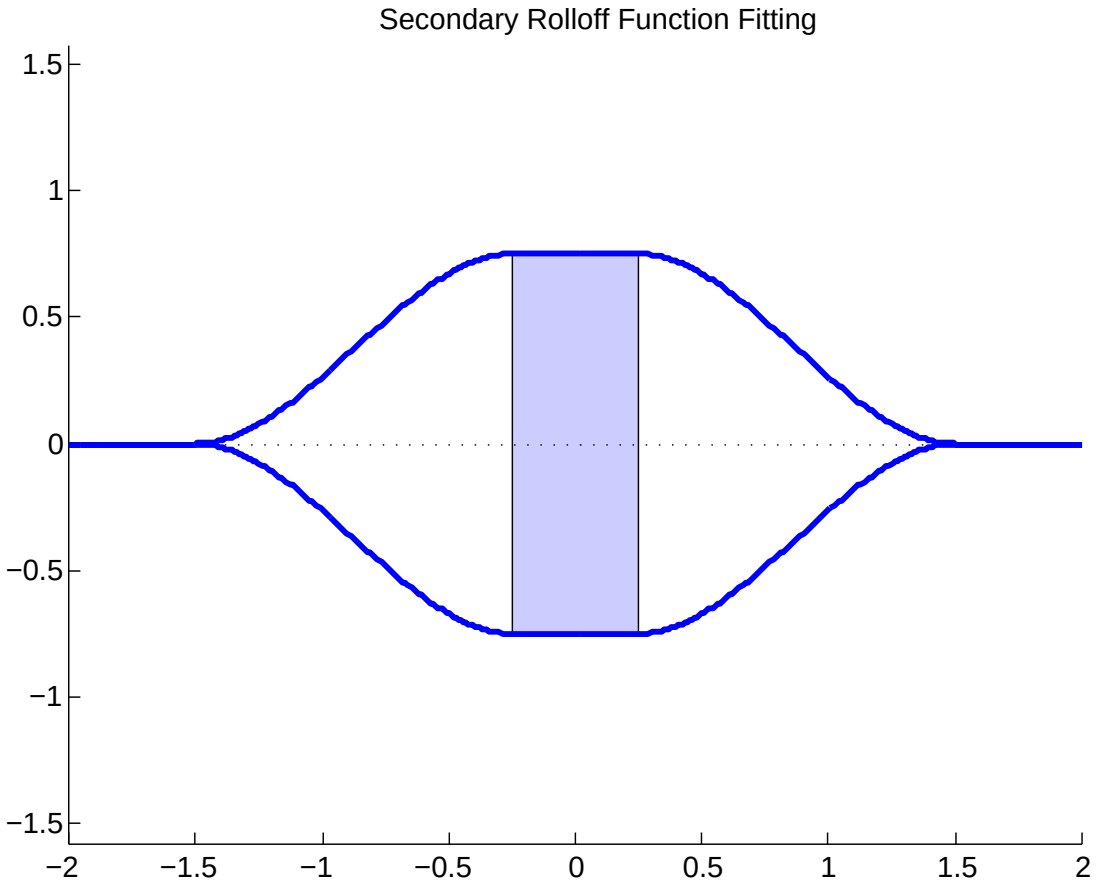

Figure 5.5. Secondary rolloff function, obstacle angle 0

goal near the side closest to the goal attractor. The orientation of the field is determined by

$$\zeta = \theta \pm \frac{\pi}{2}, \tag{5.23}$$

dependent on whether the current point is above or below the bisection line. The scalar partial field and field orientation parameters are then used to calculate the final oriented obstacle differential field,

$$\dot{q}_i = \rho \cdot \begin{bmatrix} cos(\zeta) \\ sin(\zeta) \end{bmatrix}. \tag{5.24}$$

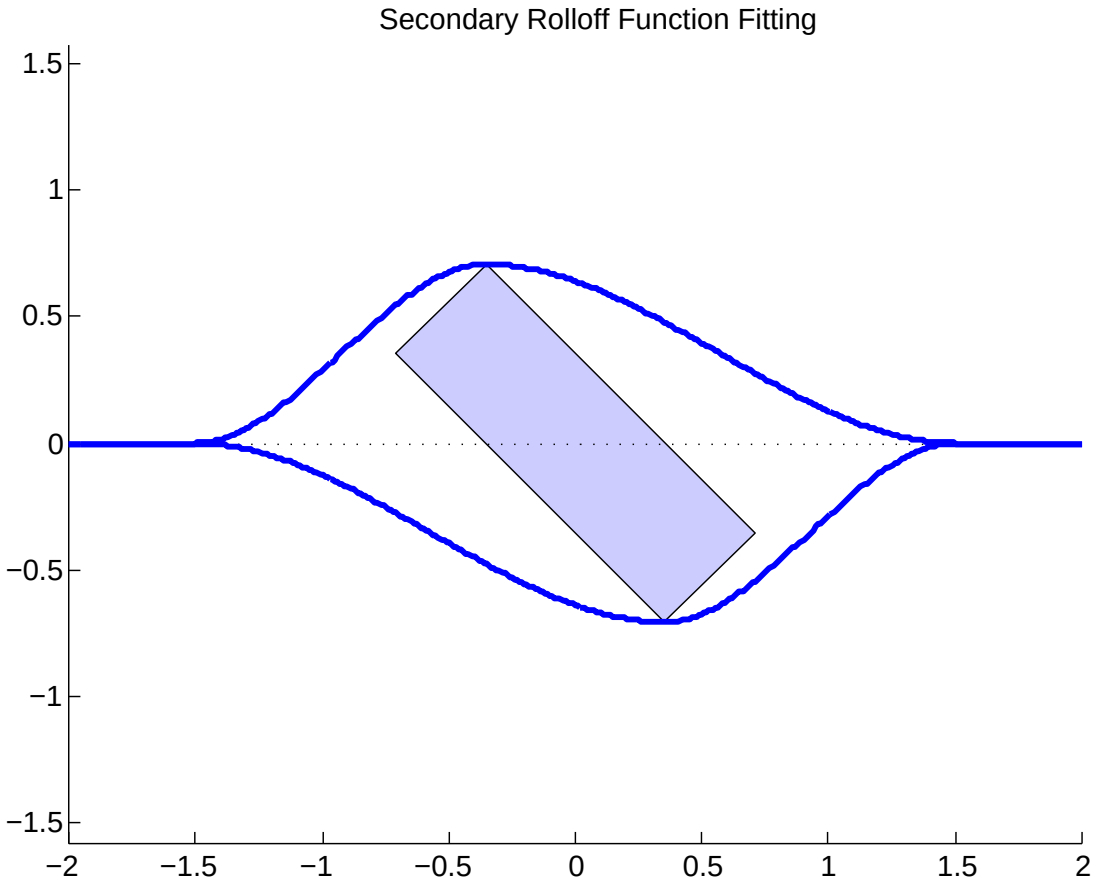

Figure 5.6. Secondary rolloff function, obstacle angle $\pi/4$

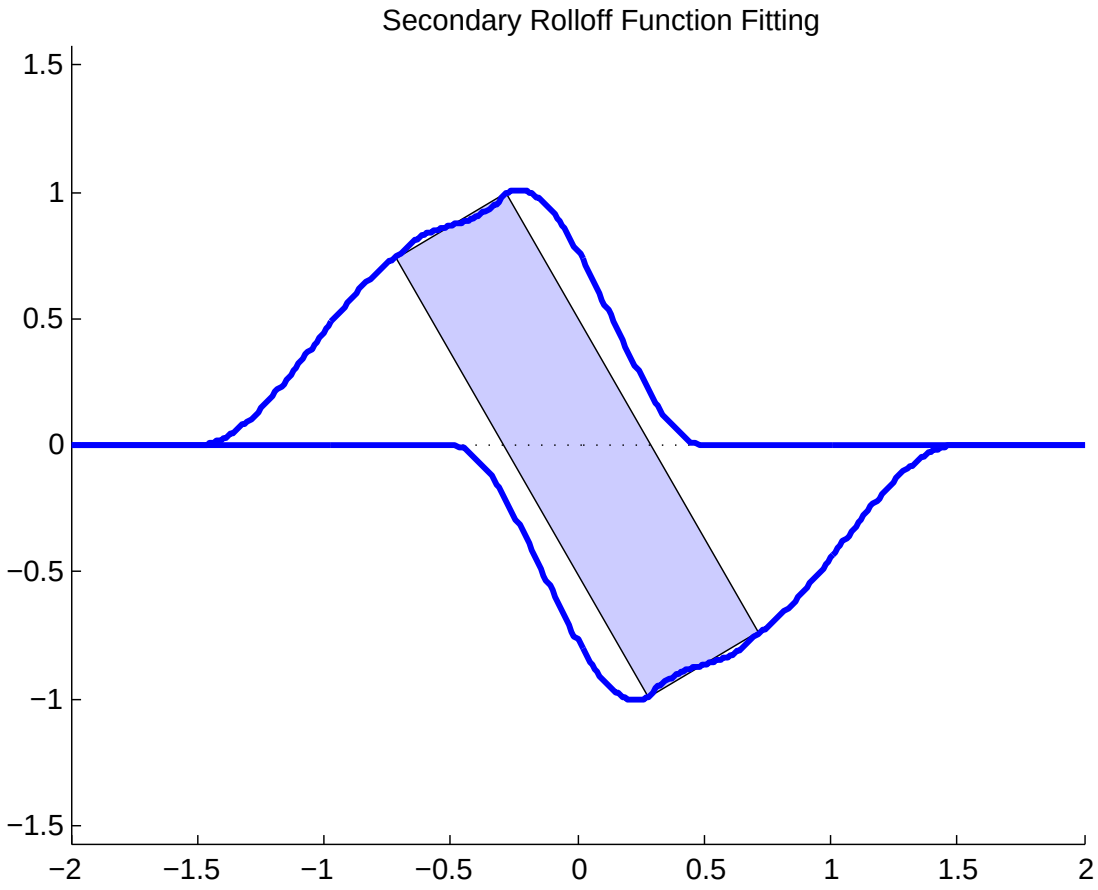

Figure 5.7. Secondary rolloff function, obstacle angle $\pi/6$

CHAPTER 6

SIMULATION

In this chapter, the design, implementation, and results of simulations for trajectory generation, motion control, and obstacle avoidance are presented. The design of all components to be implemented into Emulab Mobile are first rigorously tested in simulation to verify their characteristics and performance. All simulations presented here use MATLAB and SIMULINK.

Several simulation applications are used to evaluate the performance of the different aspects of motion planning and control. The simulations are used to speed the development of algorithms, and verify the planned features before undertaking the task of implementation in to the Emulab Mobile system. The simulations results presented in this chapter provide a baseline to establish desired behavior, which can be used to compare and verify results gathered through experimentation on real hardware.

6.1 Trajectory Generation

The trajectory generators discussed in Chapter 3 are evaluated in simulation in this section. Applications are created in the MATLAB programming language to test all aspects of the various trajectory generators before integration in to RMCD. This method is chosen to enable rapid development and evaluation of algorithms for generating segmented paths, curves, and parameterization to create final reference trajectories required by the motion controller.

For simulation and verification of the segmented trajectory generators, a standard reference trajectory is needed. A series of waypoints resulting in a path with curves with high and low angles, plus multiple segments that must be orthoganal. This standard trajectory is designed to thouroughly test all generators for robustness, and correctness. Figure 6.4 gives an example of a standard testing trajectory, colloquially termed the *double paperclip.*.

6.1.1 Line-Arc Trajectories

In Figure 6.1, the resulting paths from two trajectory generators are presented. The path created from simulation was generated from the MATLAB application. The path in the plot labeled **RMCD** was generated from the implementation on Emulab Mobile, as discussed in Section 2.5.

The velocity profiles from the two trajectory generators are compared in Figure 6.2 and Figure 6.3.

6.1.2 Spline Trajectories

An example of the standard test trajectory generated using quintic splines, as discussed in Section 3.5 is given in Figure 6.4. The fillet radius is 0.25 meters, and $a = 0.05$.

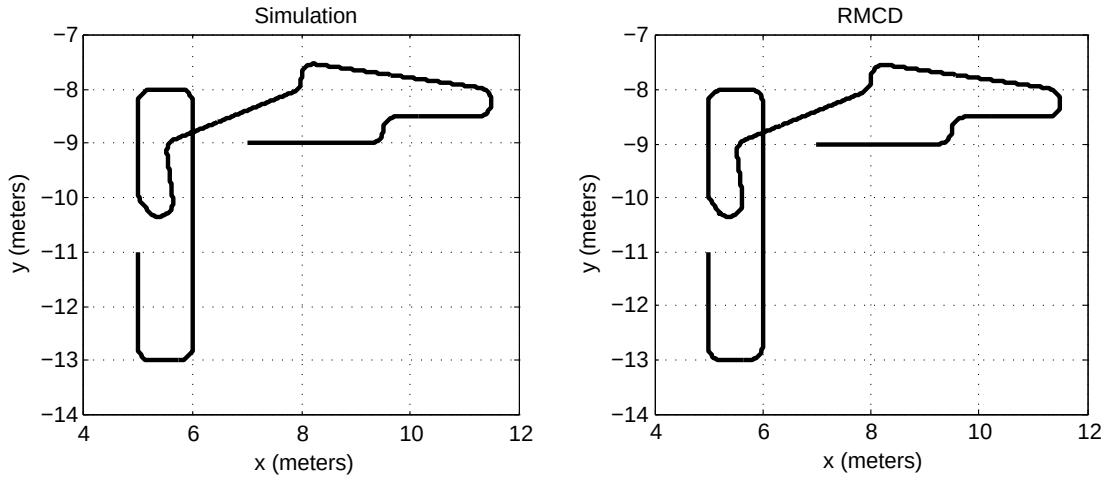

Figure 6.1. Trajectory generation comparison, path.

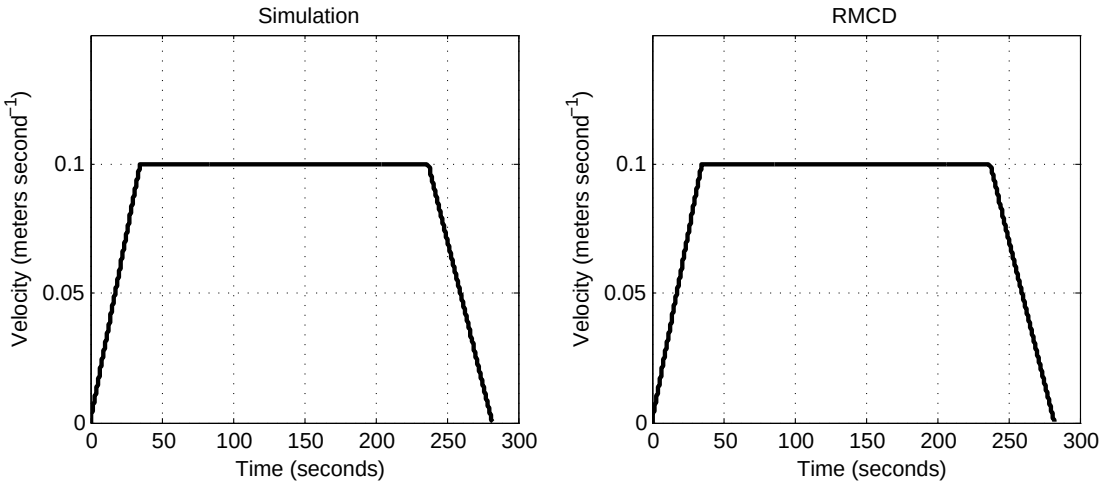

Figure 6.2. Trajectory generation comparison, velocity.

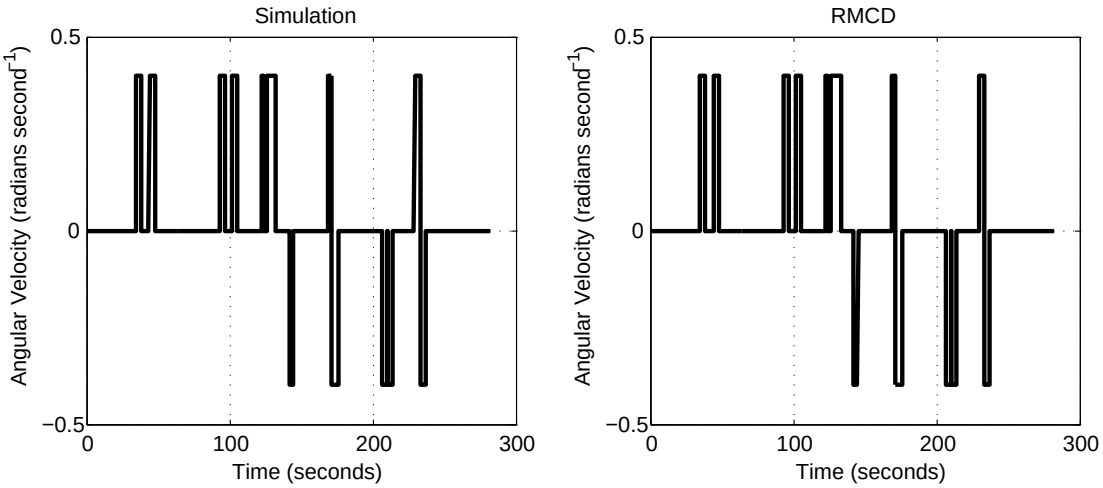

Figure 6.3. Trajectory generation comparison, angular velocity.

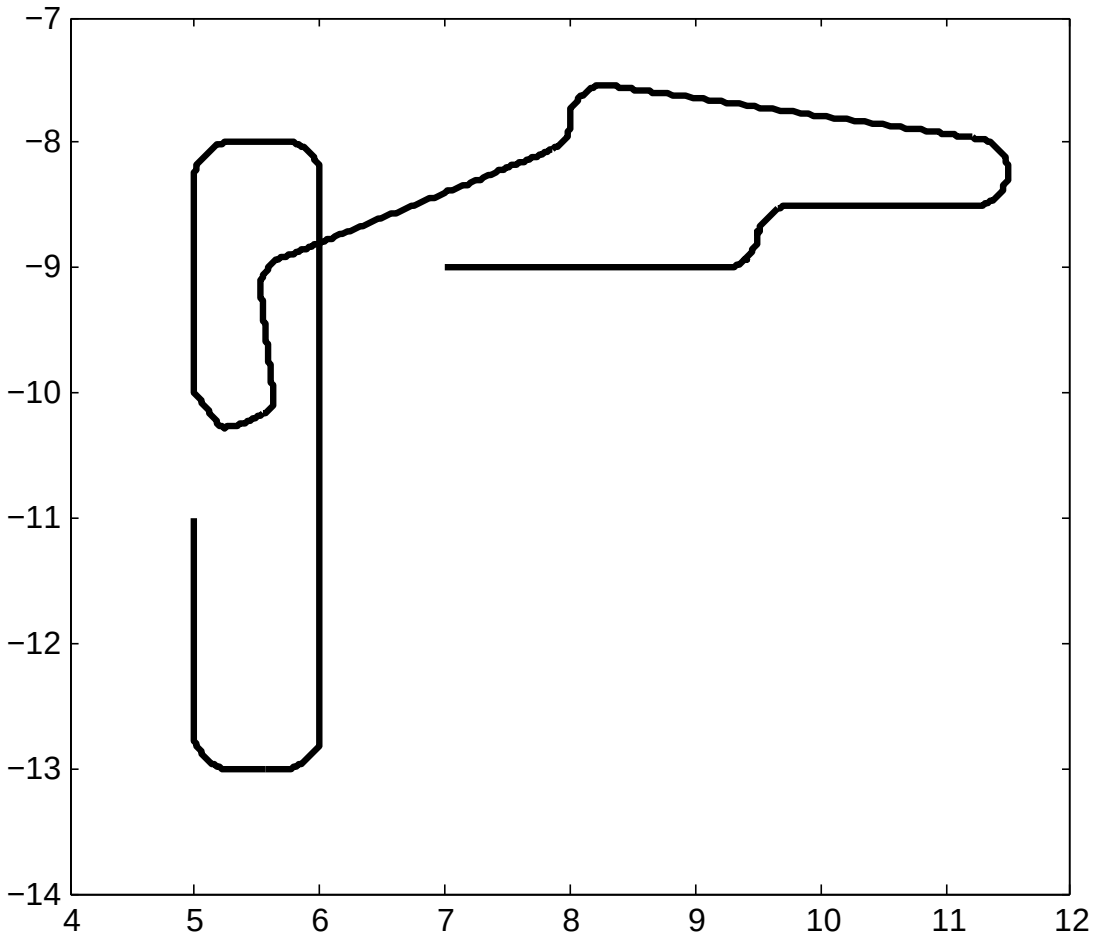

Figure 6.4. Example trajectory with quintic spline curves.

6.2 Posture Stabilization Controller

Simulations of the posture stabilization controllers discussed in Chapter 4 are created to test the stability and performance of the designs. Before integration of these controller in to Emulab Mobile, the controllers must be verified to respond as desired. The results presented in this section explore the behavior of the two posture stabilization controllers under different initial conditions, parameters, and gains. The simulations presented here do not account for the sampling frequency of the state feedback on the real system. Noise characteristics are not modeled, and robot dynamics consist mainly of the saturation velocity and acceleration of the controller command signals. The goal of these simulations is to establish the feasibility of these controllers to perform as expected while being used to control real robots.

6.2.1 Simulation Development

The posture stabilizing controller discussed in Section 4.3 is shown in Figure 6.5, as implemented in SIMULINK. Logging facilities are in place to capture position, state, controller, and wheel velocity data from the simulation. Both Cartesian and polar inputs for initial conditions are accepted.

The alternate version of the posture stabilizing controller is included in this simulation. This version allows only forward motion, avoiding cusps in the resulting paths. The block diagram for this controller is given in Figure 6.6.

A SIMULINK block diagram of the main simulation application of the posture stabi-

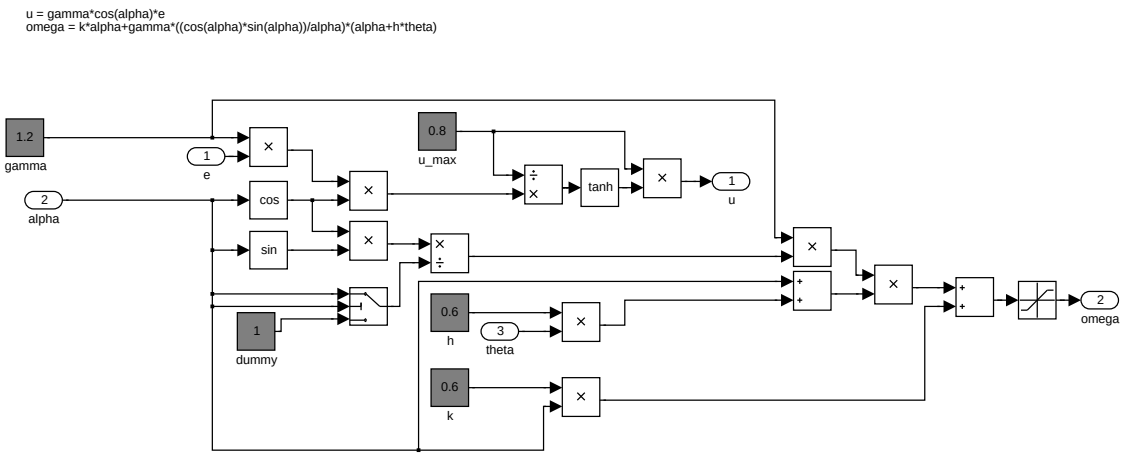

Figure 6.5. Posture stabilizing controller.

lizing controller is shown in Figure 6.7. Included in this application are blocks to accept initial conditions in either Polar or Cartesian coordinates, and output of the resulting trajectory, Polar states, and controller output velocities. Both posture regulators are included, and can be chosen at run time.

The robot polar kinematics block diagram, shown in Figure 6.8, takes input from the controller, runs through a robot dynamics model, and then directly implements the polar state equations, (4.3). The initial state values come from the values set in the main simulation.

6.2.2 Simulation Results

Figure 6.9 shows the paths generated by the posture stabilizing controller discussed in Section 4.3. Initial postures are located at various angles on a unit circle. The initial angle is zero at each instance. For initial postures with a large value of the polar state θ, the resultant paths include cusps, where the linear velocity changes directions. Two simulations are presented here, both with an initial postures resulting in higher values of θ. These postures were chosen since they result in more interesting trajectories. All postures are presented in the form $[xy\phi]$. Unless otherwise noted, linear measurements are in meters, and angular units are radians.

The alternate posture stabilizing controller is simulated with an initial posture of

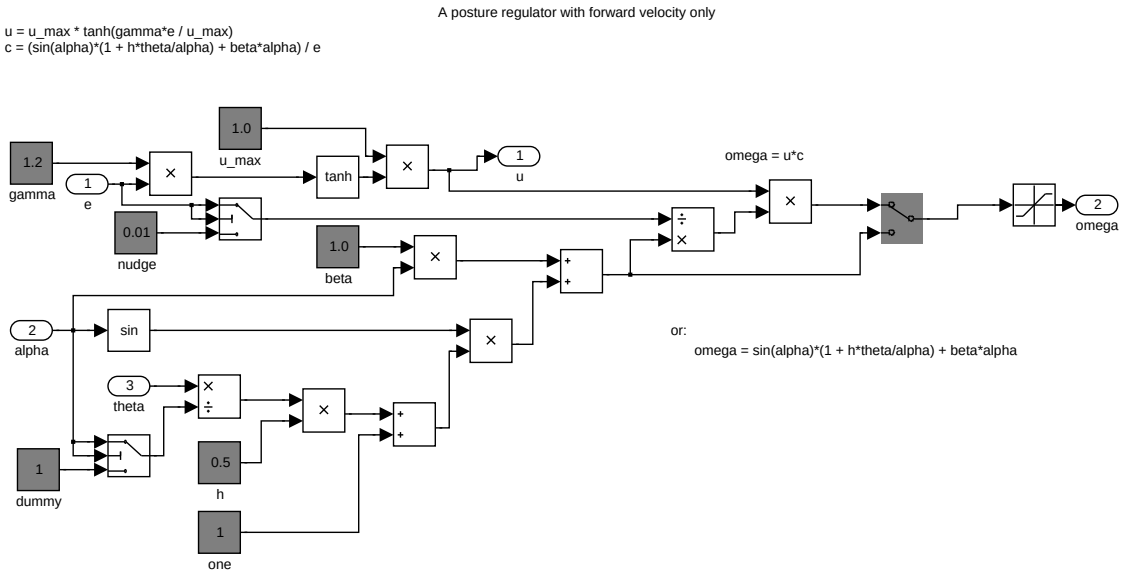

Figure 6.6. Posture stabilizing controller, alternate version.

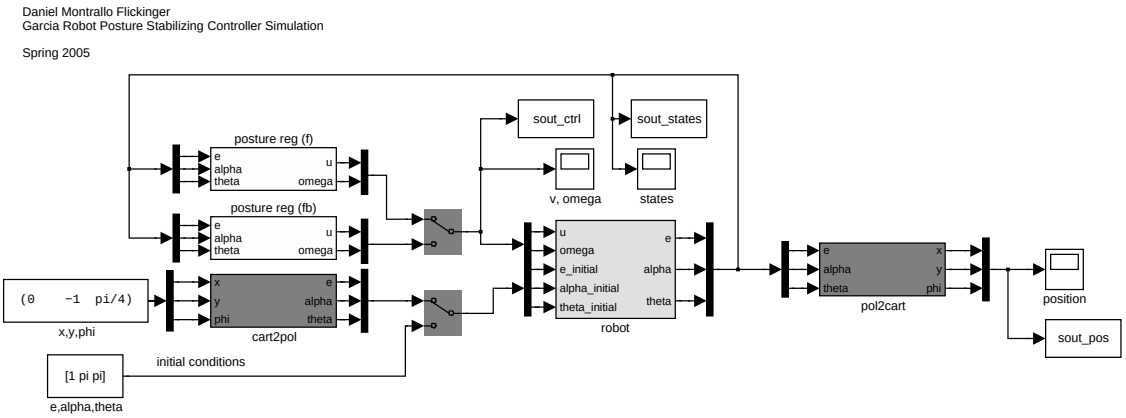

Figure 6.7. Posture stabilizing controller: main simulation application.

$[x, y, \phi] = [0.0, -1.0, -\pi/4]$. The results of this simulation are shown in Figure 6.10. The light gray structures represent the posture of a differentially steered robot's axle at constant time intervals. This posture stabilizing controll is designed to command forward motion only, avoiding any cusps in the resulting path.

The polar system states of this simulation are given in Figure 6.10. As desired, all states are regulated to zero, with no overshoot. The controller is demonstrated here to be exponentially stable for this specific set of initial conditions and controller parameters. Figure 6.12 displays plots of the controller outputs.

The simulated wheel velocities are presented in Figure 6.13, with the corresponding wheel accelerations given in Figure 6.14.

The main posture stabilizing controller is simulated with initial Cartesian states of $[xy\phi] = [-0.6 - 1.2 - \pi/2]$. The resulting trajectory is given in Figure 6.15. These initial conditions result in a cusp in the path. The robot is initially oriented South in this figure. This results in the robot undergoing backward motion over the first section of the trajectory.

The simulated Polar system states are plotted in Figure 6.16. All states smoothly progress to zero, and settle rapidly.

Output from the controller is given in Figure 6.17, with the resulting wheel speeds in Figure 6.18. The wheel velocities differ from the controller command because of acceleration saturation.

The wheel accelerations are shown in Figure 6.19. These accelerations are saturated at 0.4 meters per second squared.

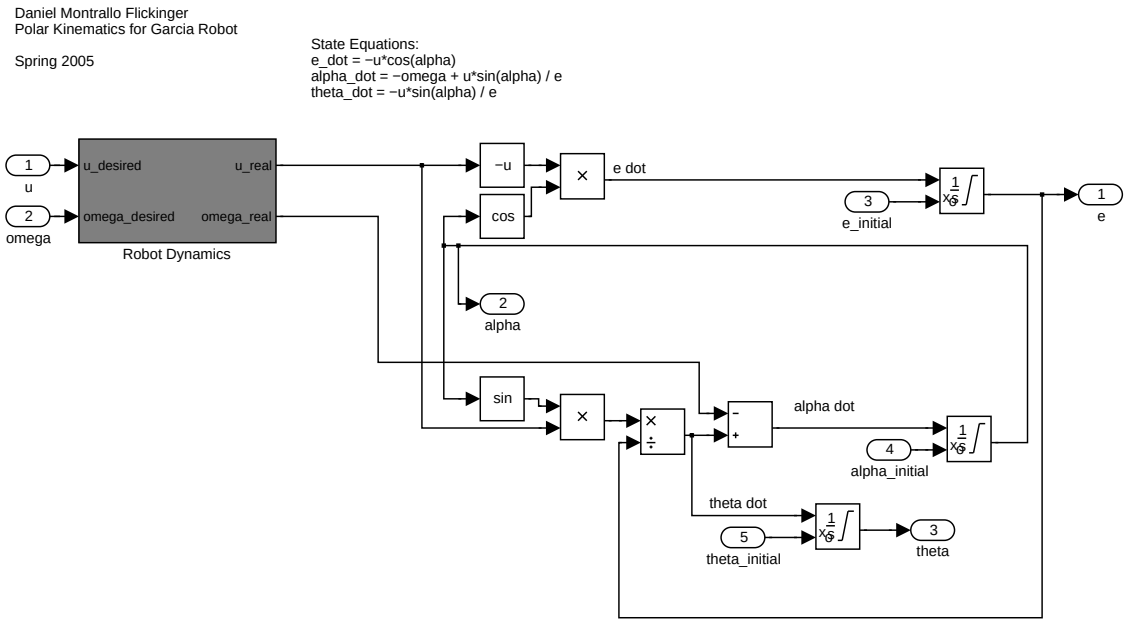

Figure 6.8. Robot polar kinematics simulation block diagram.

The simulations presented in this section establish that the posture stabilizing controller presented in Section 4.3 is exponentially stable, and produces bounded and smooth output. With continuous state feedback, kinematic control may be used to drive a robot to a single goal posture. In the next section, a kinematic trajectory tracking controller is tested in simulation. The development of the posture regulator serves as an initial effort, and establishes the kinematics and control structure required to proceed to development of more advanced motion controllers.

6.3 Kinematic State Feedback Trajectory Tracking Controller

The trajectory tracking controller presented in Chapter 4 is evaluated in simulation in this section. This controller requires careful simulation design, and rigorous testing, because of the challenges presented by the state feedback sampling rate and other aspects of the Emulab Mobile system. The goal in this section is obtain results that establish the feasibility of using this controller to control actual hardware.

An example trajectory generated by the motion controller discussed in Section 4.4 is shown in Figure 6.20. This simulation is an initial result from the controller, with little gain adjustment, and a continuous sampling rate. The reference trajectory is the

standard *double paperclip* trajectory, as presented in Section 6.1. The simulated trajectory initially converges slowly, but eventually manages to track the reference path closely for the duration of the simulation.

Figure 6.21 illustrates the initial convergence of a simulated trajectory to a reference trajectory. The initial position is approximately five centimeters offset from the start of the reference trajectory.

6.3.1 Simulation Development

The main simulation block diagram for the kinematic state feedback trajectory tracking controller simulation is included in Figure 6.22. This simulation is completed in MATLAB and SIMULINK. It includes input of a reference trajectory, and logging of states, gains, controller outputs, and trajectories to and from the MATLAB workspace. Polar states are calculated from Cartesian trajectories according to (4.3). The motion

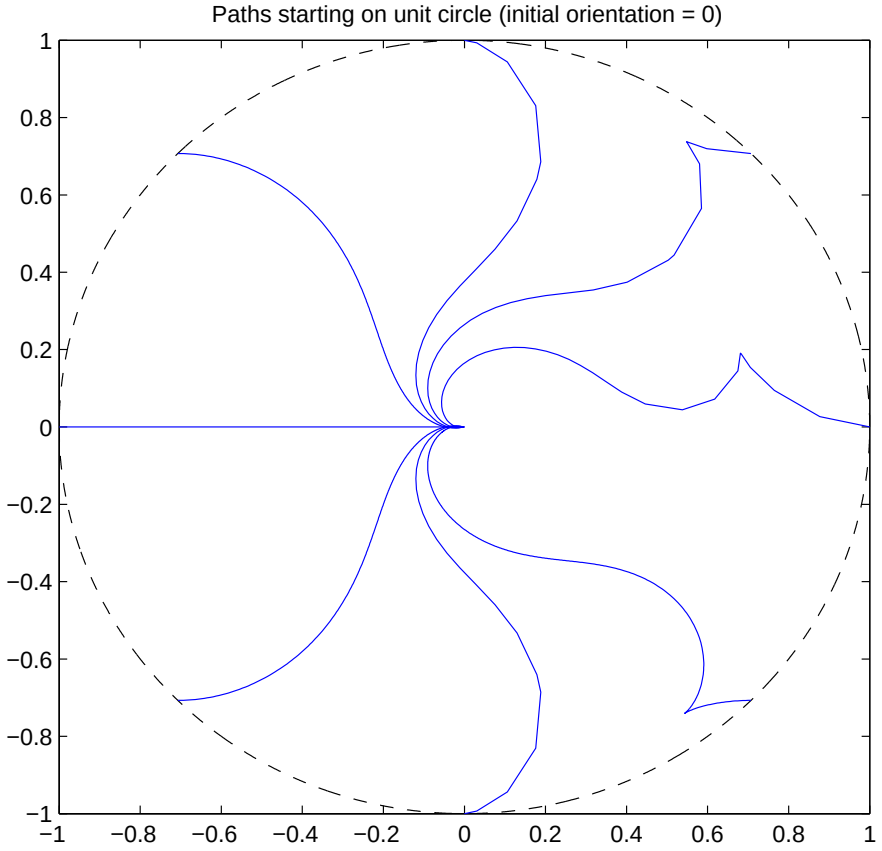

Figure 6.9. Posture stabilizing controller simulation: paths resulting from initial postures on a unit circle.

controller output is passed to a block which calculates the resulting trajectory using the Cartesian state equations, (4.1).

Figure 6.23 presents the block diagram of the core controller and dynamic extension. The control laws of these components are (4.39), (4.42) and (4.43), (4.44) respectively.

The models in simulation are designed to account for acceleration and velocity saturation on the real robots. The sampling frequency is also considered by applying a zero order hold with a rate of 30 Hz to the dynamic extension output. Facilities are built into the simulation to log all state and localization data for creating plots for later review.

Applications are written to noninteractively run the controller system simulation with varying parameters. Discrete derivatives are used to best model the RMCD implementation. The simulation is run at a constant sampling rate of one kilohertz.

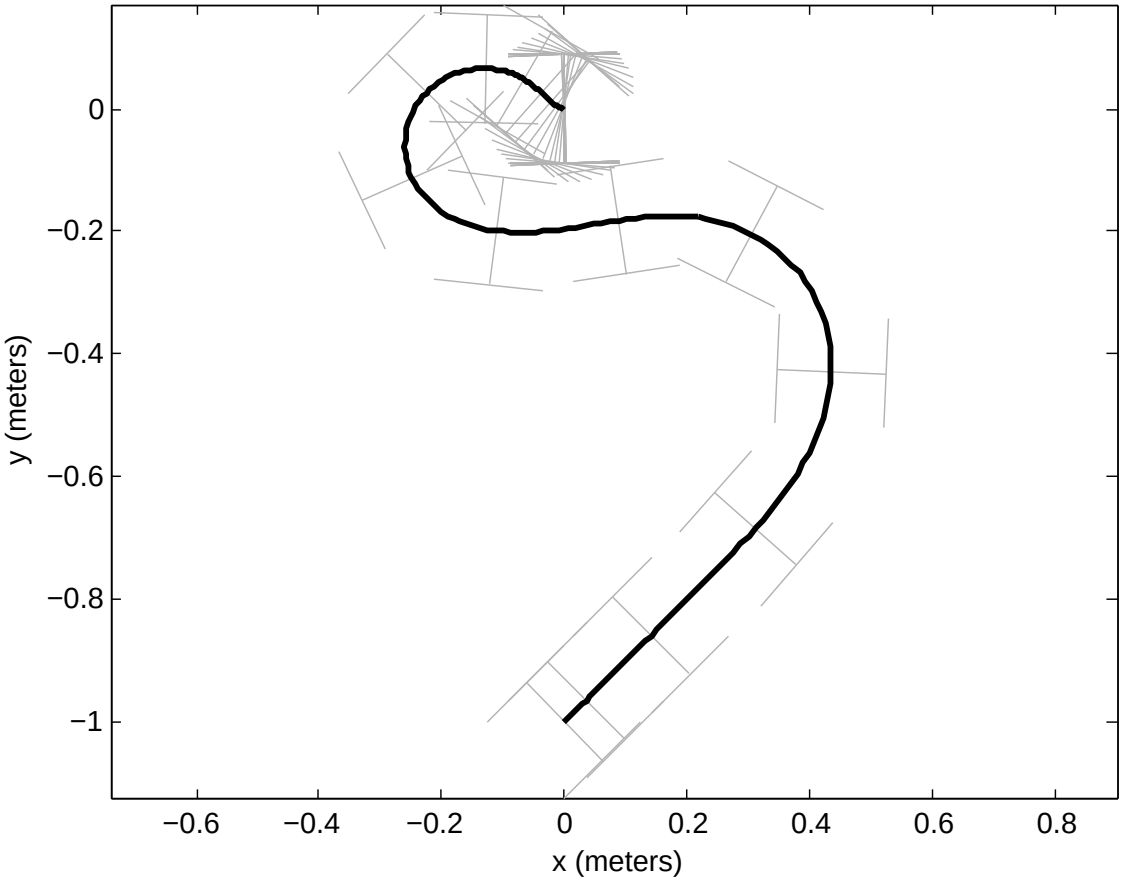

Figure 6.10. Posture stabilizing controller, simulated trajectory.

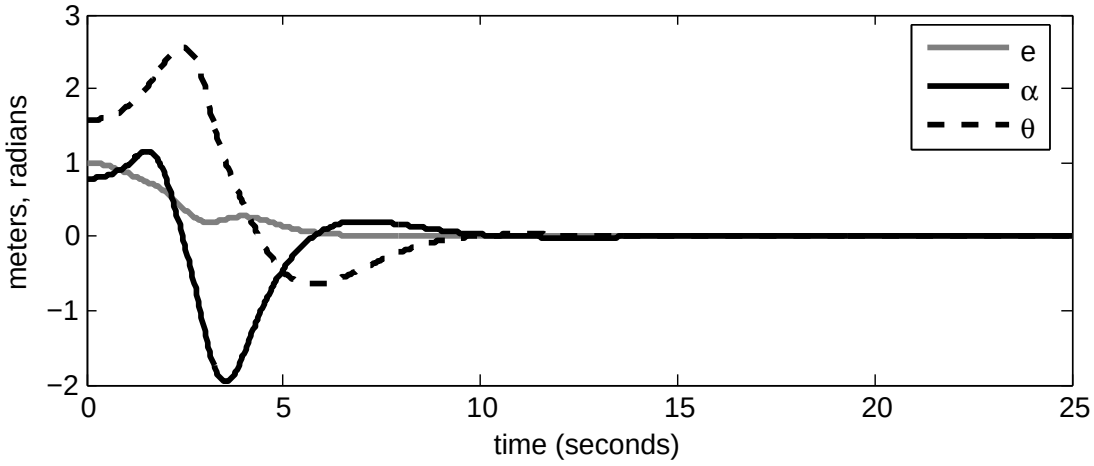

Figure 6.11. Posture stabilizing controller, simulated system response, corresponding to Figure 6.10.

6.3.2 Initial Conditions

An initial position which is 'ahead' of the start of a reference trajectory may cause problems, as shown in Figure 6.24. The control law given in Subsection 4.4.1 does not allow backward motion. A consequence of this is that simulated trajectories will loop around to join the reference trajectory, instead of backing up or pausing until the reference trajectory has caught up. This sort of trajectory would be difficult to send as a command to a real robot, as the velocities and accelerations required are too high.

A series of tests are performed to determine safe initial conditions for the motion controller. A sample of the evaluations are presented here. All postures presented here are in the form $[xy\phi]$. The initial trajectory convergence is shown, and it may be assumed that the out of boundary trajectories resemble the trajectories presented in Figure 6.20. The *double paperclip* reference trajectory starts at $[xy\phi] = \left[5.0 - 11.0 - \frac{\pi}{2}\right]$.

Figure 6.25 has a favorable initial posture of $\left[4.95 - 10.95 \frac{-\pi}{2}\right]$. The simulated trajectory converges rapidly, with no overshoot or oscillations. The initial conditions are favorable in this case because the position results in the initial absolute value of the polar kinematic state θ to be less than $\frac{\pi}{2}$ radians. The orientation of the initial posture also matches the initial orientation of the reference trajectory.

The simulated trajectory shown in Figure 6.26 is similar to Figure 6.25, except that the initial posture lies on the opposite side of the reference path. The state θ has switched signs, but is still less than $\frac{\pi}{2}$ radians. The trajectory convergence is roughly identical to

the previous simulation. The initial posture in this simulation is $\left[5.04 - 10.95 - \frac{\pi}{2}\right]$.

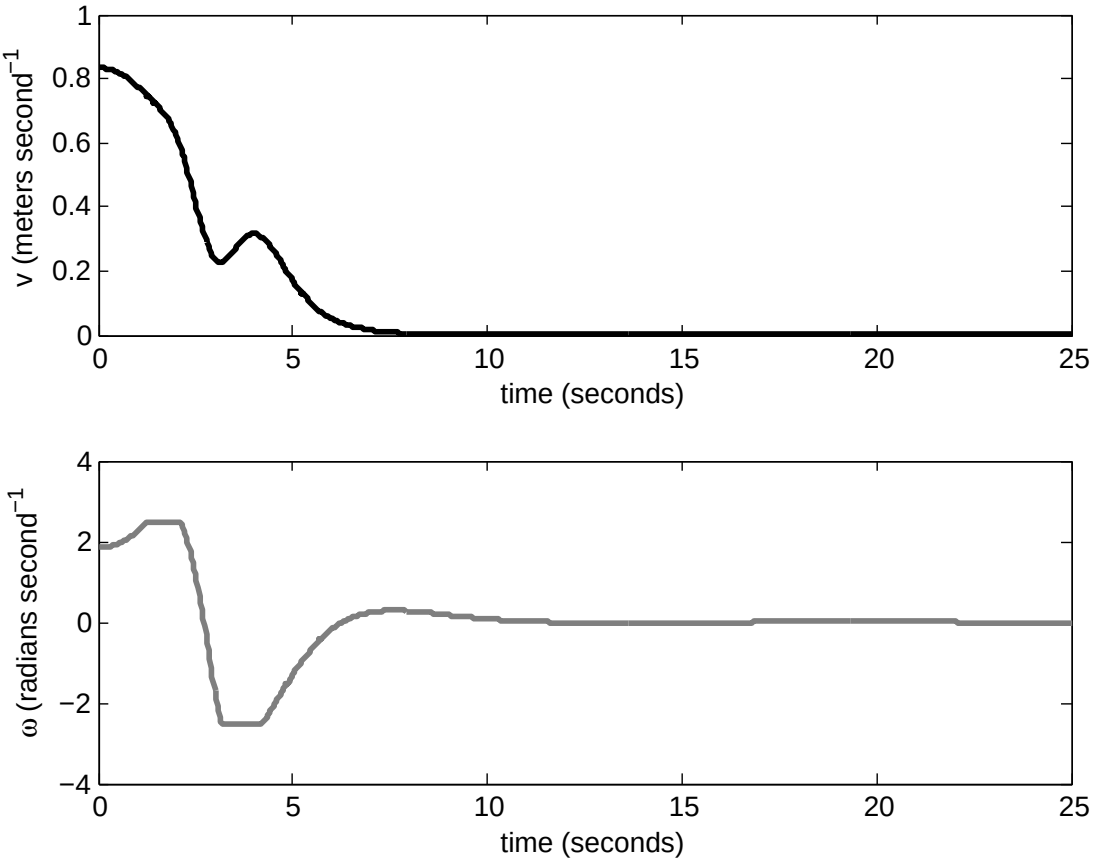

Figure 6.12. Posture stabilizing controller, controller output.

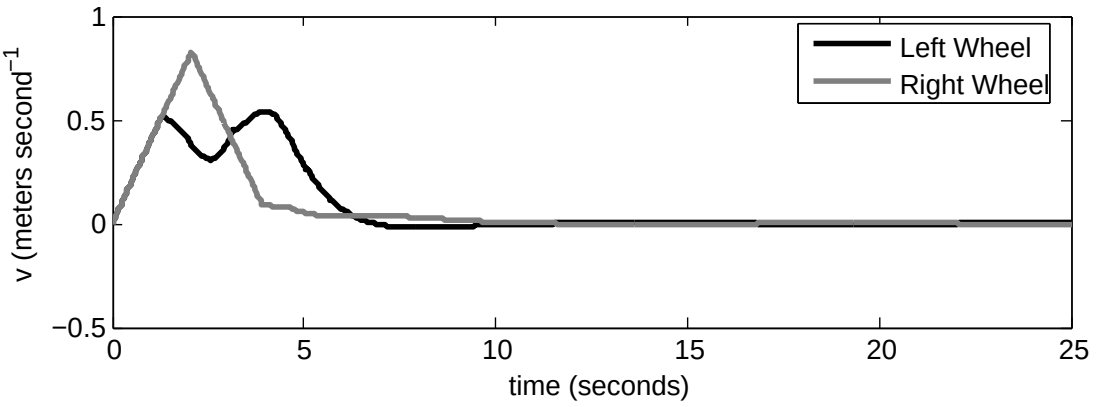

Figure 6.13. Posture stabilizing controller, wheel velocities.

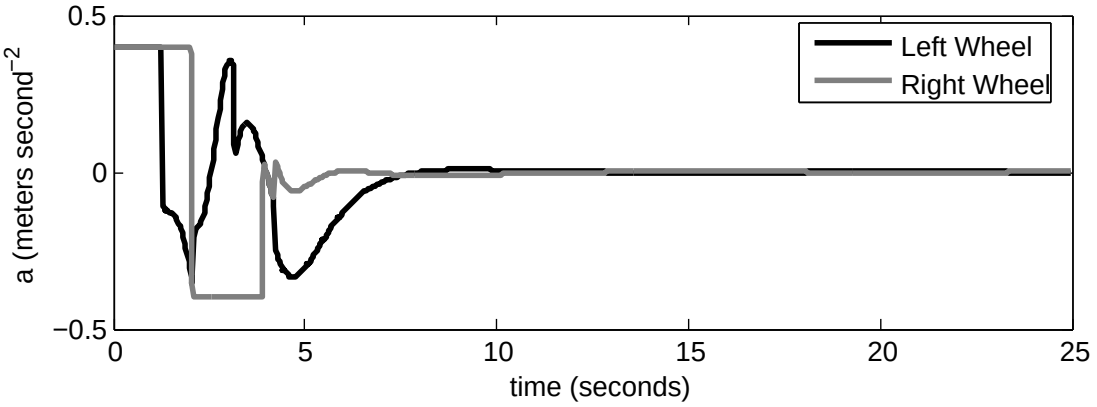

Figure 6.14. Posture stabilizing controller, wheel accelerations.

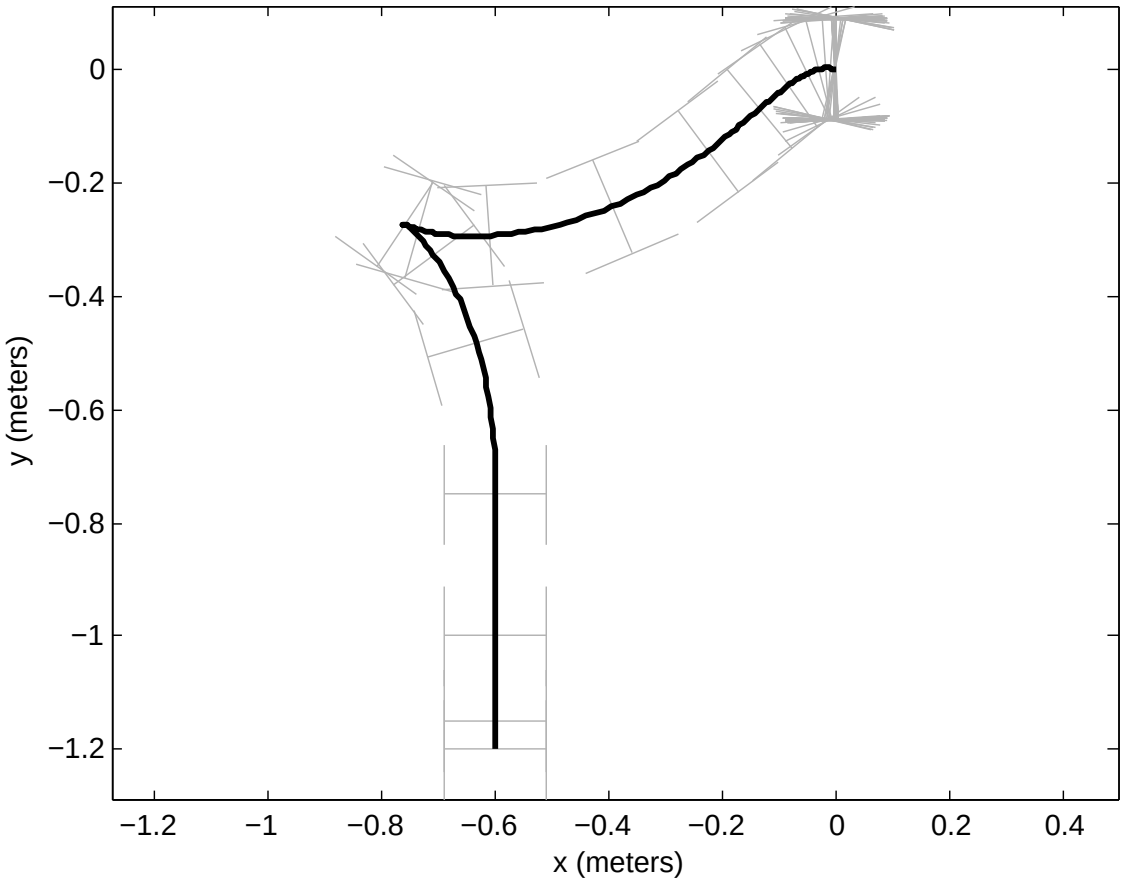

Figure 6.15. Posture stabilizing controller, $[-0.6 - 1.2 - \pi/2]$, simulated trajectory.

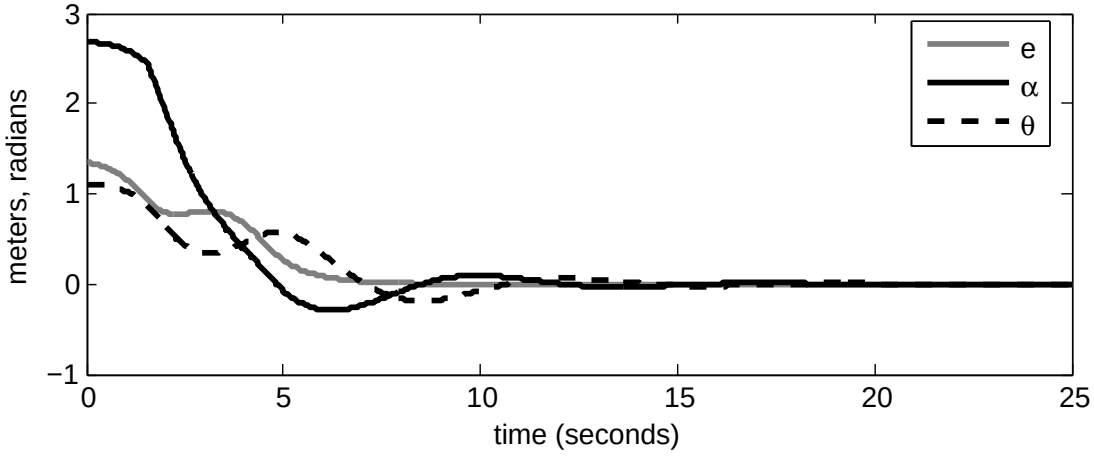

Figure 6.16. Posture stabilizing controller, $[-0.6 - 1.2 - \pi/2]$, simulated system response.

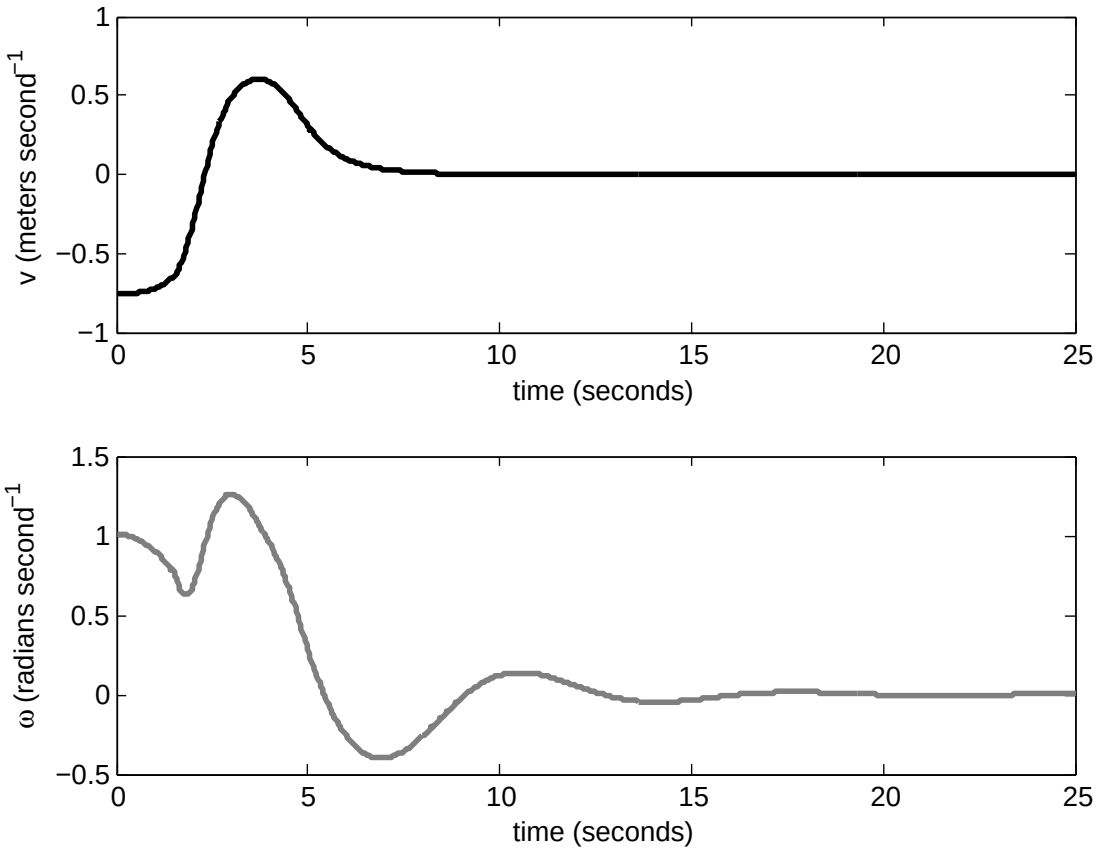

Figure 6.17. Posture stabilizing controller, $[-0.6 - 1.2 - \pi/2]$, controller output.

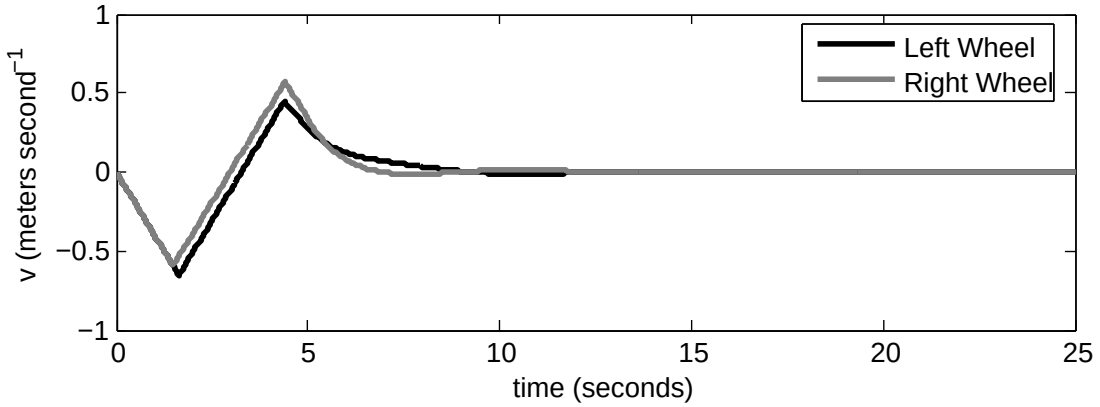

Figure 6.18. Posture stabilizing controller, $[-0.6 - 1.2 - \pi/2]$, wheel velocities.

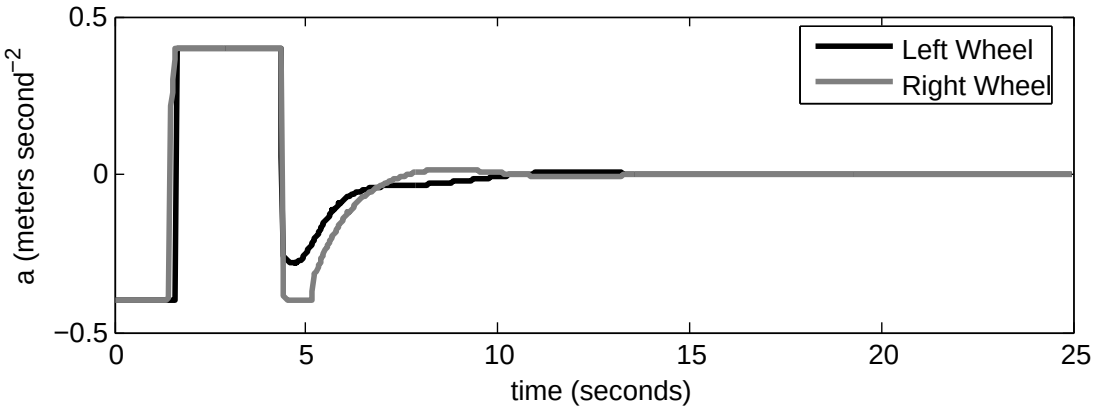

Figure 6.19. Posture stabilizing controller, $[-0.6 - 1.2 - \pi/2]$, wheel accelerations.

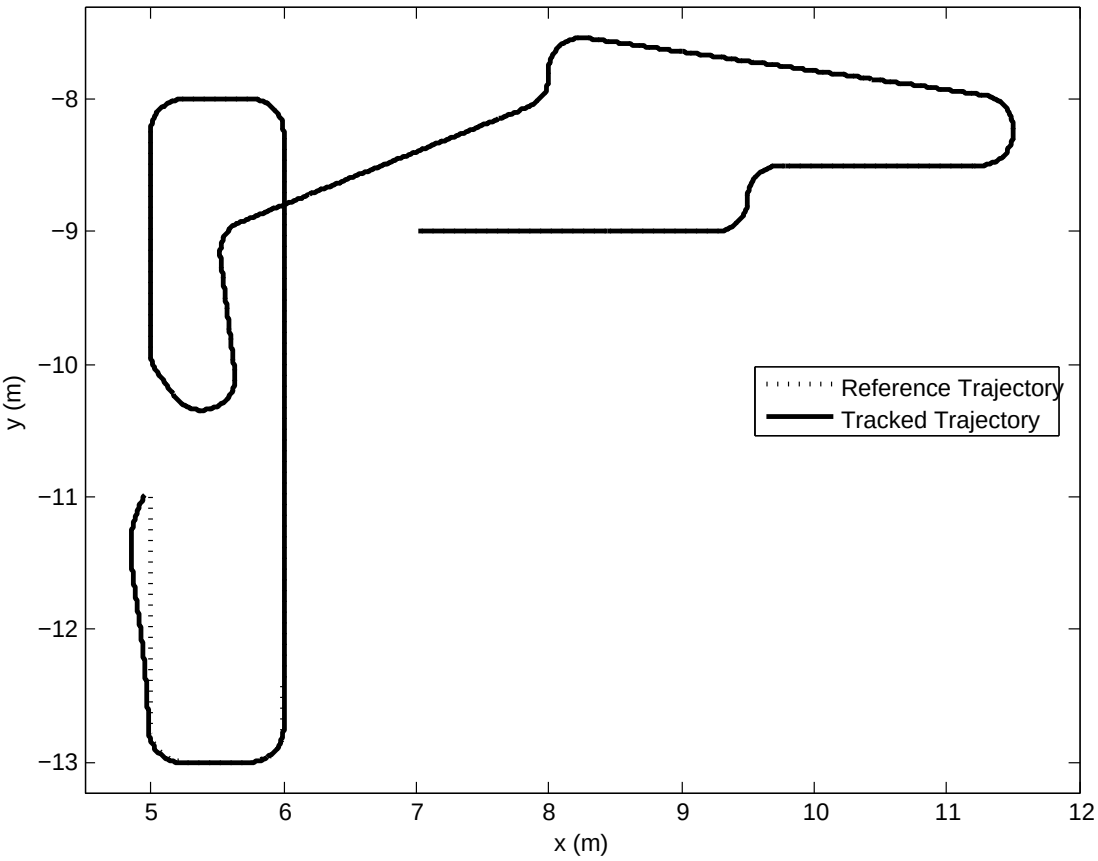

Figure 6.20. Trajectory tracking controller, continuous sampling.

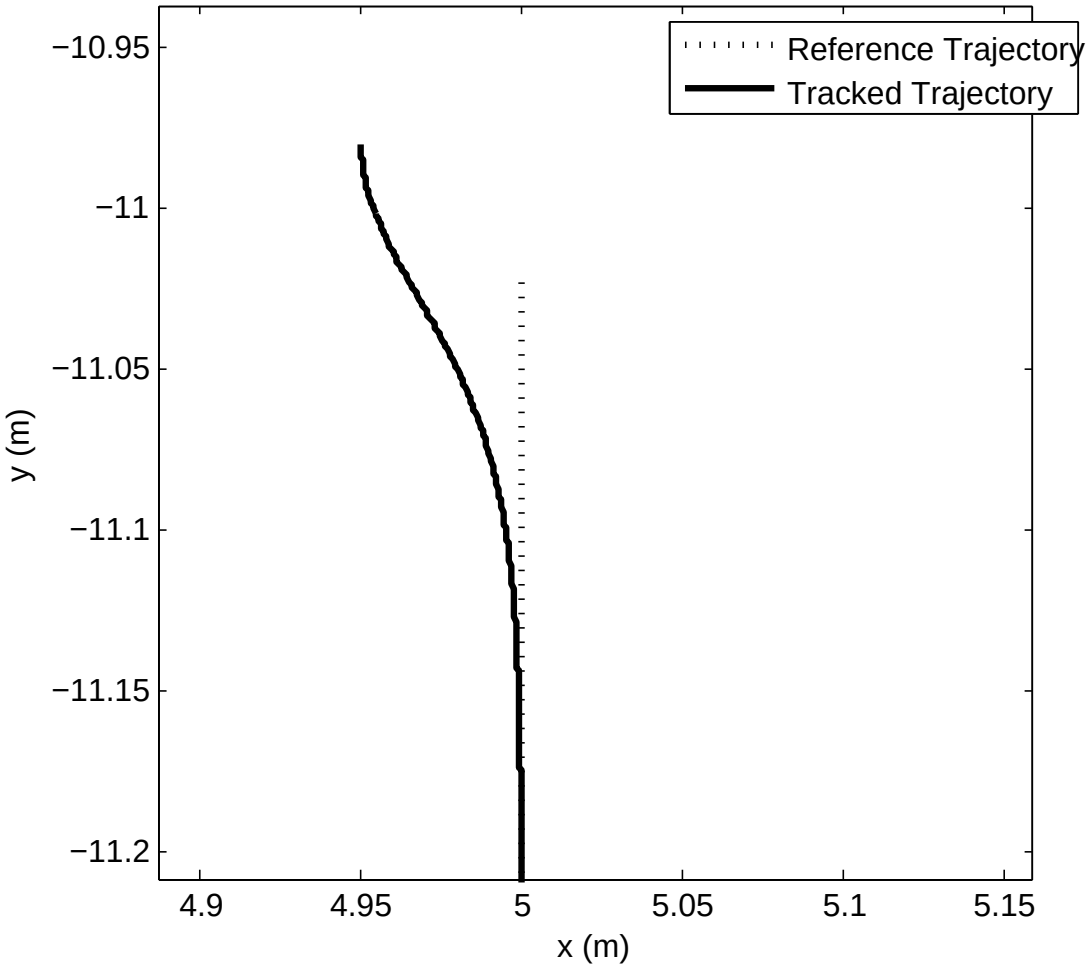

Figure 6.21. Trajectory tracking controller, initial path convergence.

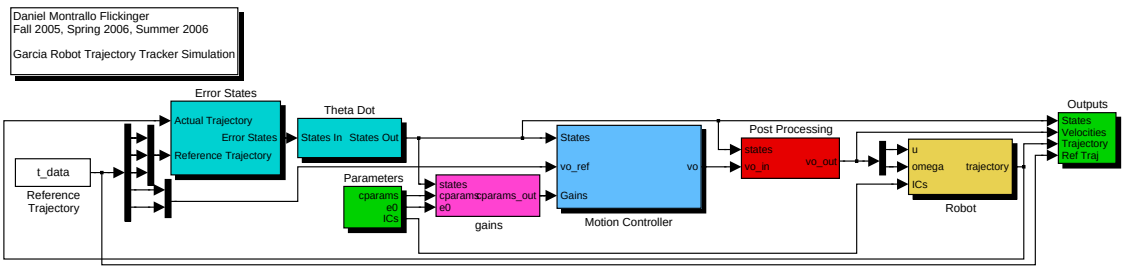

Figure 6.22. Trajectory tracking controller simulation block diagram.

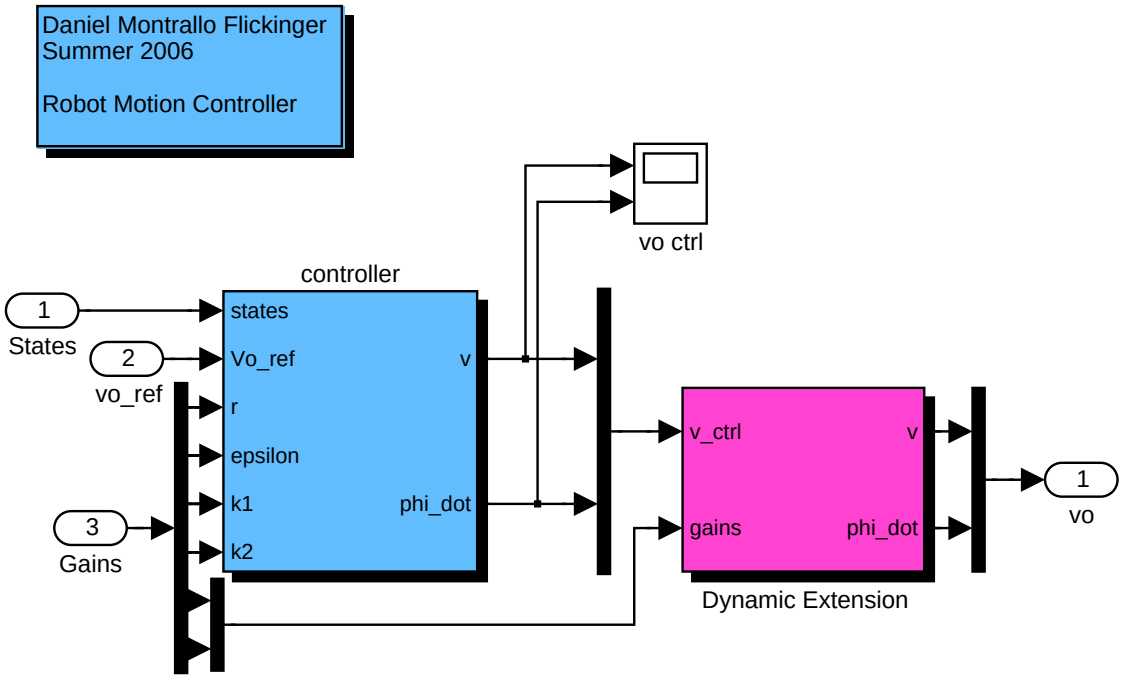

Figure 6.23. Trajectory tracking controller simulation, motion controller block diagram.

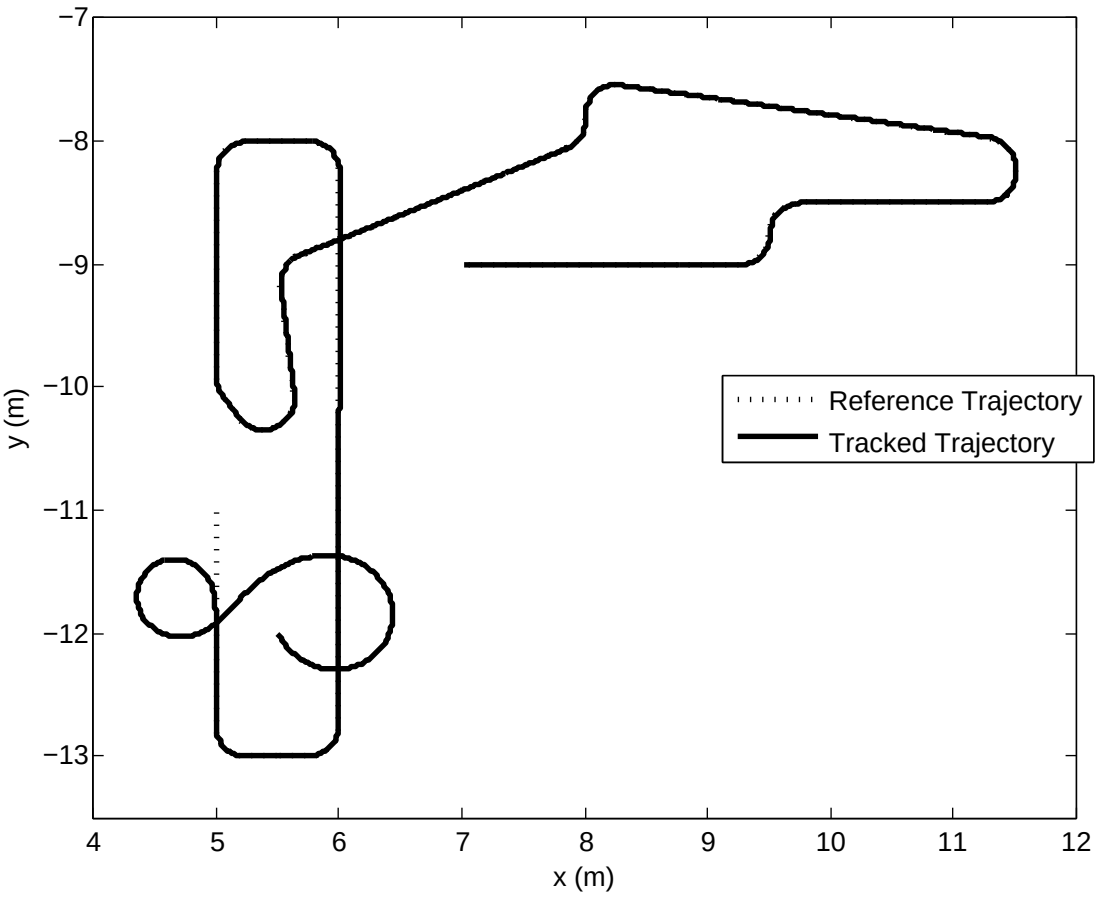

Figure 6.24. Path loop caused by initial conditions.

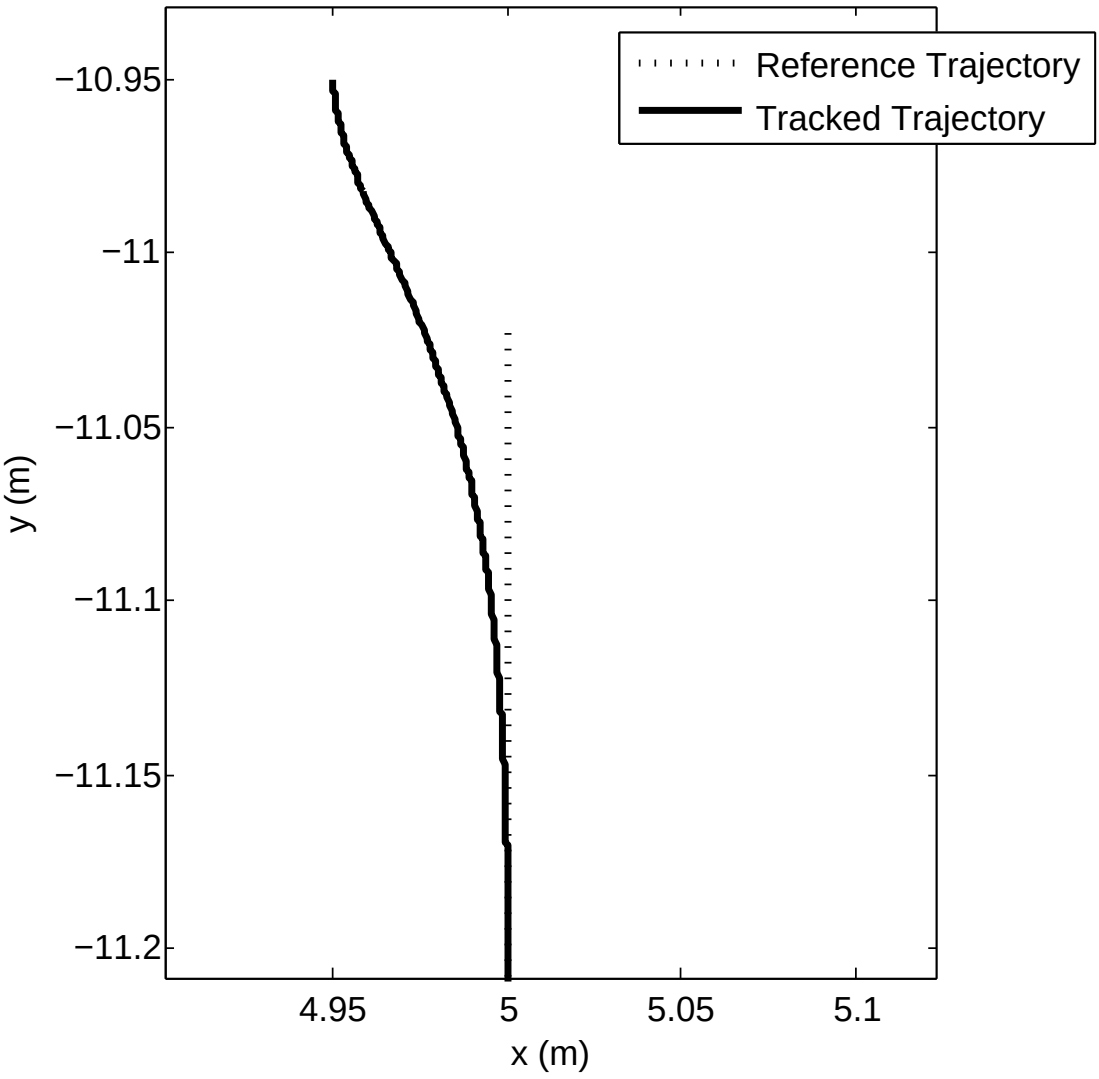

Figure 6.25. Initial trajectory with boundary conditions $\left[4.95 - 10.95 \frac{-\pi}{2}\right]$.

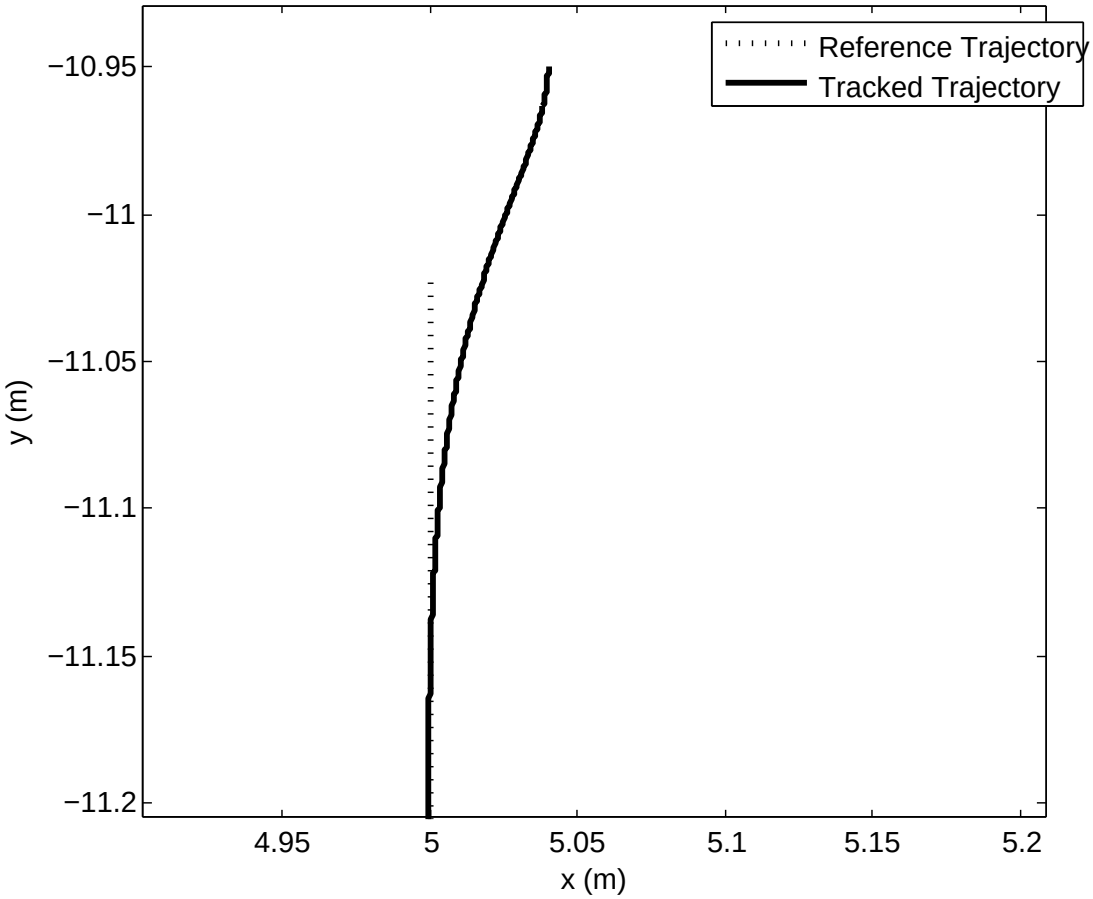

Figure 6.26. Initial trajectory with boundary conditions $[5.04\ -10.95\ \frac{-\pi}{2}]$.

An example of a poorly formed initial posture is given in Figure 6.27. An initial posture of $[5.04 - 10.99 - \pi]$ is used in this simulation. The initial position is satisfactory, but the orientation is pointed away from the initial reference posture. This results in a high value of the kinematic polar state α. To converge upon the reference trajectory, the motion controller commands a large fast loop, which results in large error initially.

A high initial value of α does not necessarily cause problems, as evidenced in Figure 6.28. The initial posture is $\left[4.99 - 11.03 - \frac{\pi}{3}\right]$. The trajectory converges rapidly, in a similar fashion to more ideal boundary conditions.

6.3.3 Sampling Rates

Simulating the motion controller under conditions approximating an ideal continuous sampling rate establishes the feasibility and performance of trajectory tracking using

this design, but the implementation constraints dictate that state feedback will only be available at a known discrete sampling rate. As established by the localization system hardware, state data is available at a maximum rate of 30 Hz. An example of a simulated trajectory generated initially after quantizing the controller output at 30 Hz is given in Figure 6.29. Without gain adjustment, and well formed initial conditions, the controller quickly fails to track the reference trajectory. This is especially a problem when encountering discontinuities in curvature at the endpoints of arcs in the path. No saturation limits on controller output are used in this simulation. The resulting trajectory loops many times, as the simulated robot continually overruns the reference trajectory.

The sample rate of state feedback may diminish as network communications drop out. In order to prepare the motion controller for use with sampled feedback, simulations are run with the controller output quantized at sampling rates between 10 Hz and 30 Hz. The resulting trajectories are compared to the motion controller simulated with continuous output.

To evaluate controller performance at varying sample rates, the overall time average of the absolute position error is measured. Absolute position error is a term for the kinematic state e from (4.3). The results of the controller simulated at different sample rates is shown in Figure 6.30. The vertical line in the center of the plot is located at a sample rate of 30 Hz. Error increases significantly as sampling frequency decreases, and the controller time step increases. In this set of simulations, a sampling rate of approximately 1 kHz is required to achieve a desired average error of one centimeter. At 30 Hz, average error exceeds three centimeters. This error would be acceptable for Emulab Mobile.

Figure 6.31 shows a simulated trajectory with a continuous controller. The controller output is quantized at 30 Hz, and the results presented in Figure 6.32. There are no saturation limits on the accelerations or velocities of the controller output in the simulations in this subsection. With unbounded velocity and acceleration, the controller can track the reference trajectory at a slow sampling rate. A few fast loops are present, but overall, the controller minimizes error.

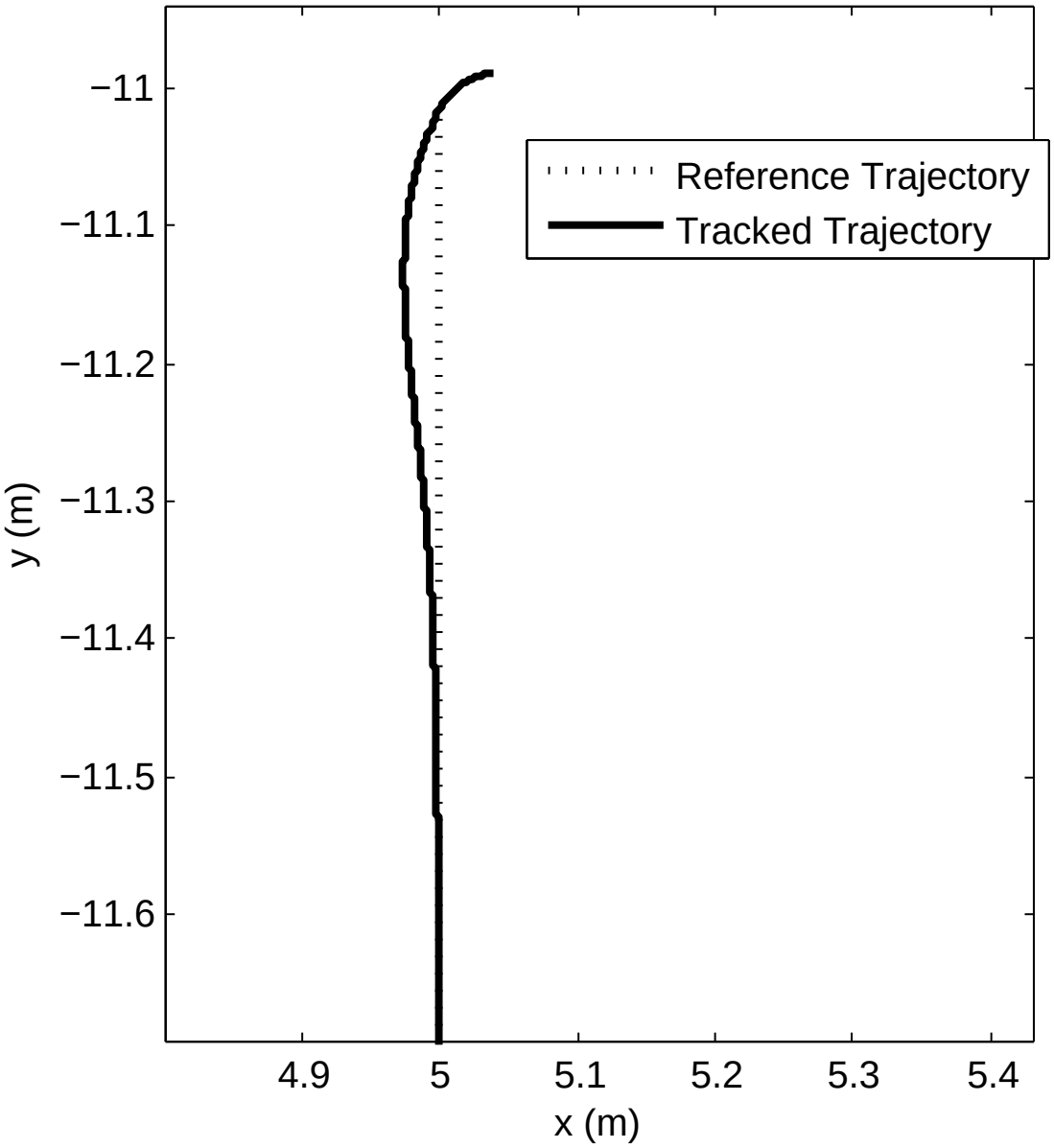

Figure 6.27. Initial trajectory with boundary conditions $[5.04\ -10.99\ -\pi]$.

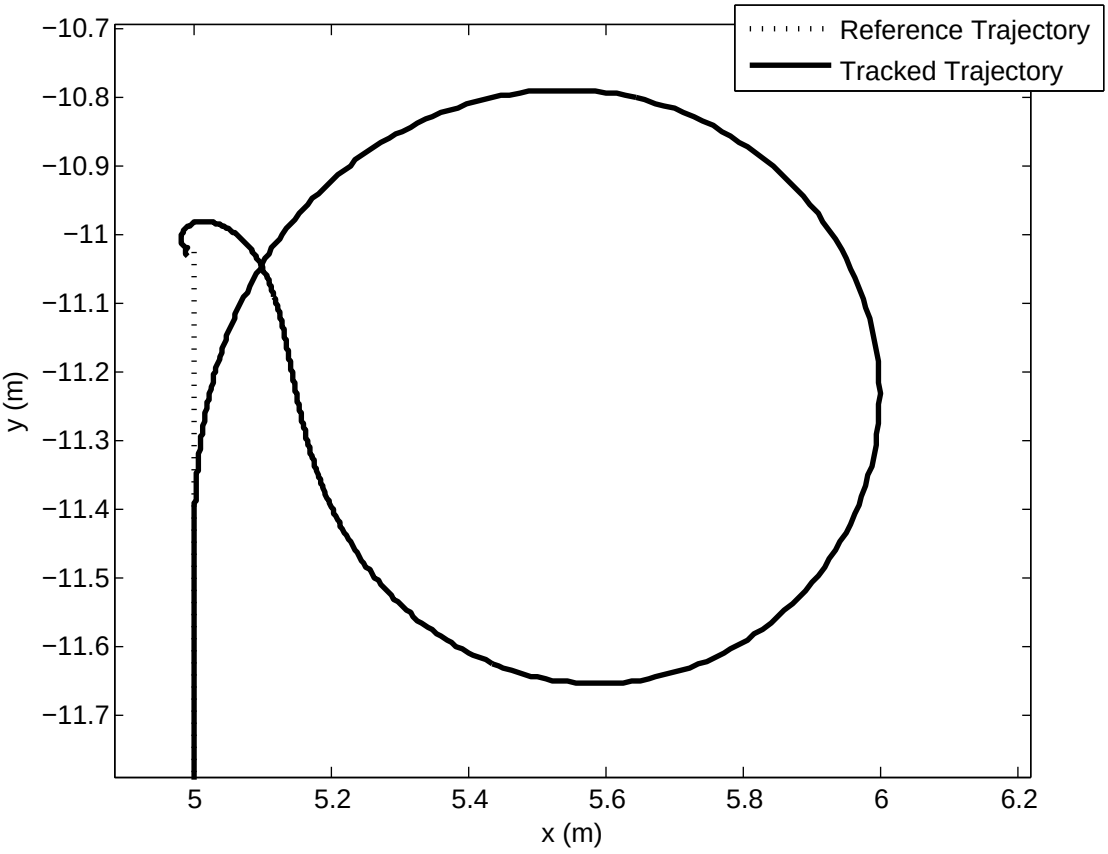

Figure 6.28. Initial trajectory with boundary conditions $[4.99\ -11.03\ \frac{-\pi}{3}]$.

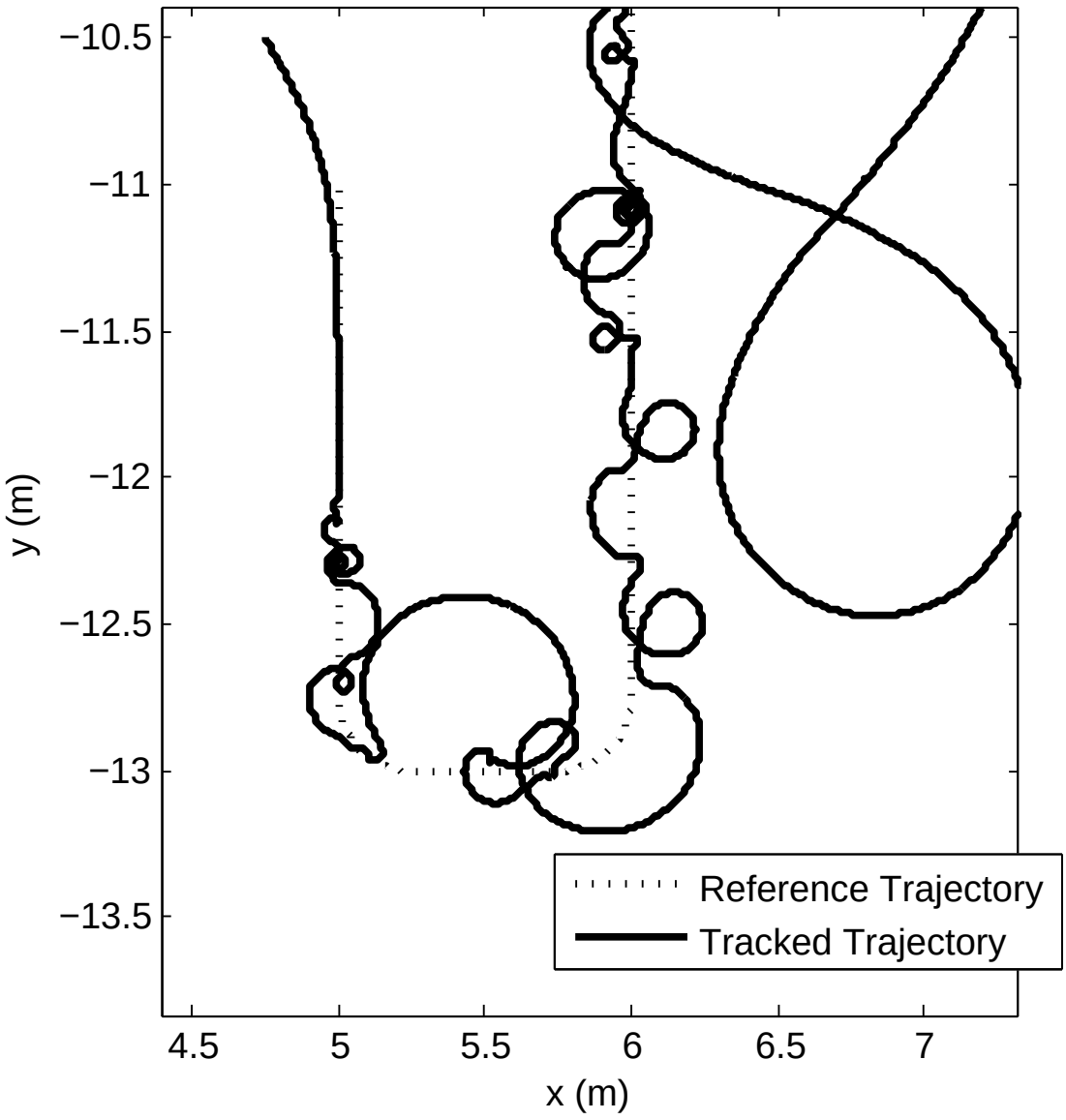

Figure 6.29. Initial simulation with controller output sampled at 30 Hz.

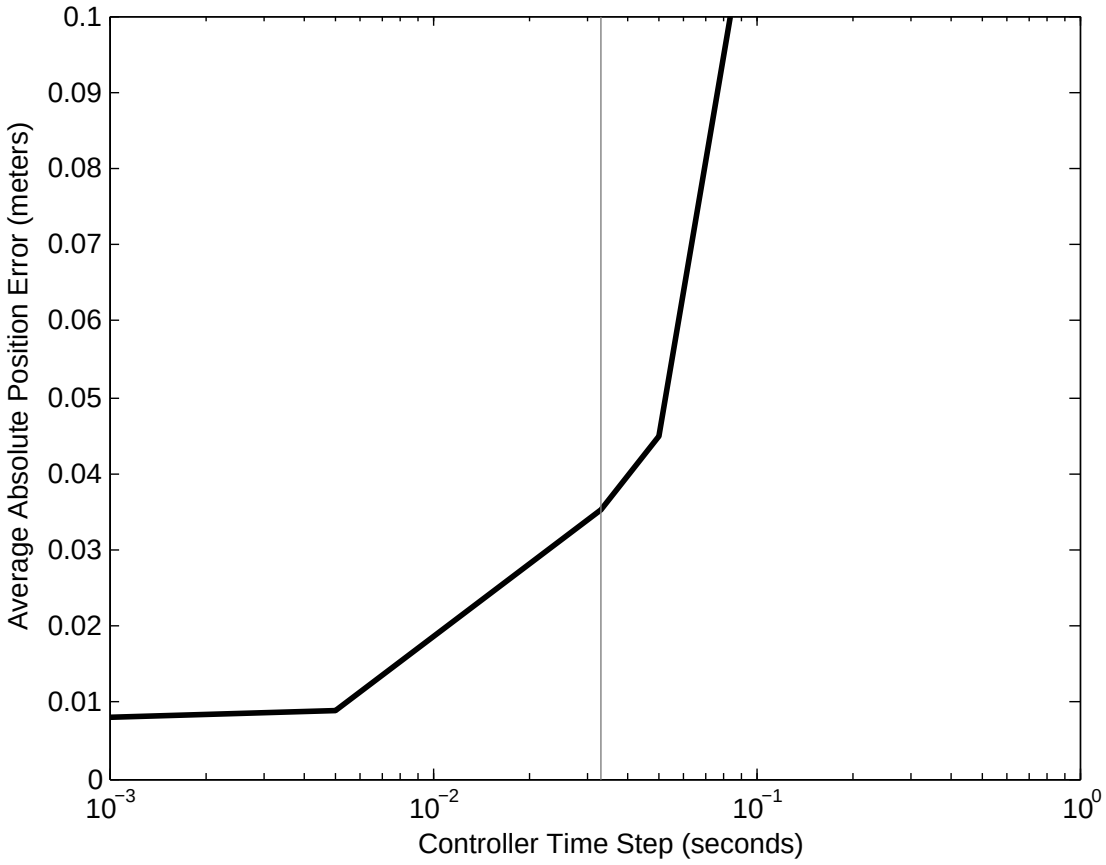

Figure 6.30. Controller performance under varying sampling rates.

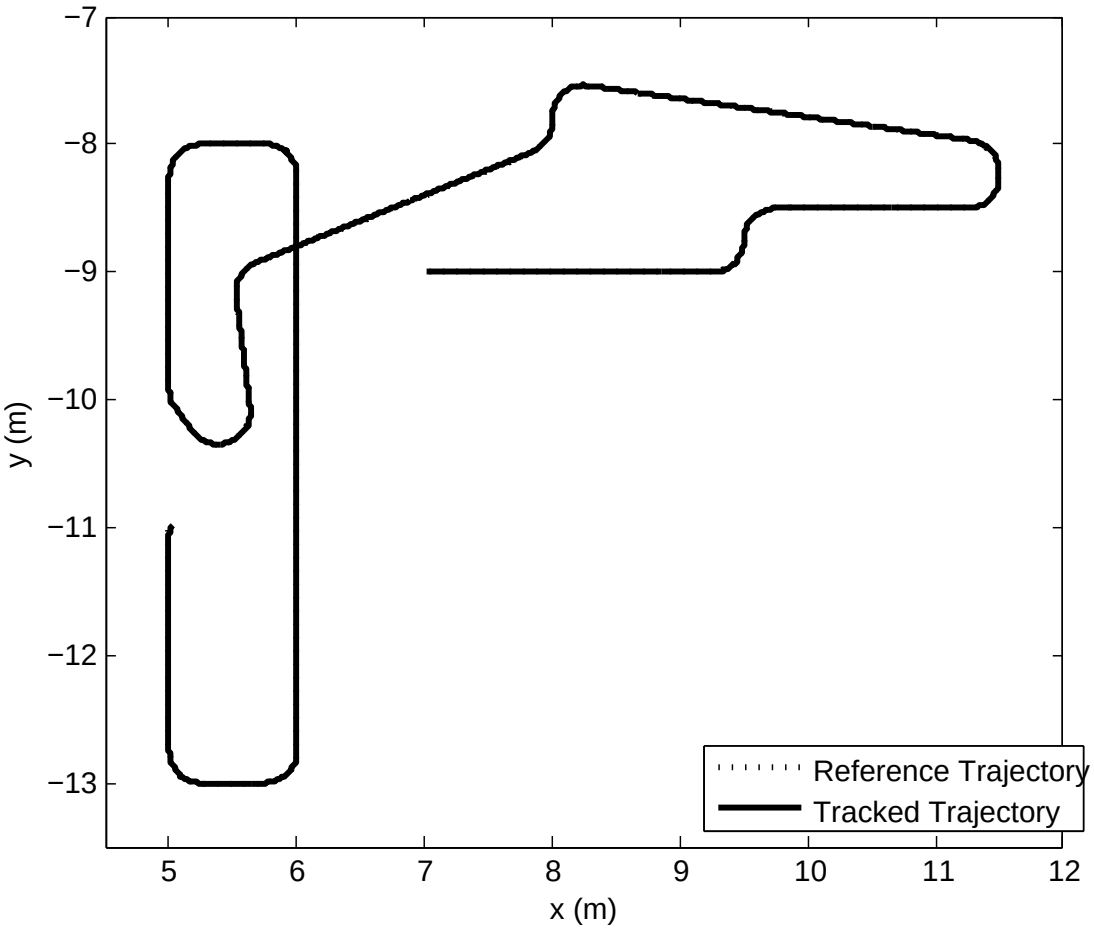

Figure 6.31. Simulation with continuous controller (no sampling).

The sampling rate is decreased to 20 Hz. Figure 6.33 gives the results of this simulation. More loops are present compared to Figure 6.32, but error is still minimized. Decreasing the sampling rate further results in Figure 6.34. THe controller becomes marginally stable, though it still tracks the reference trajectory without significant error.

6.3.4 Discrete System Stability Analysis

The trajectory tracking controller system is discretized, and then analyzed to choose parameters that result in a stable system. The system sampling rate of 30 Hz is considered in the discrete system. Parameters for k_1, k_2, k_v and k_c are chosen based on stability, and desired values for r and ϵ. Desired z transform roots are chosen based upon pole placement.

The system state equations in Polar form are defined as,

$$f_(x) = \begin{bmatrix} -v_a \cos(\alpha) + v_r \cos(\theta) \\ \frac{v_a \sin(\alpha)}{e} - \frac{v_r \sin(\theta)}{e} \\ -\omega_r, \frac{v_a \sin(\alpha)}{e} - \frac{v_r \sin(\theta)}{e} - \omega_a \end{bmatrix}. \tag{6.1}$$

The parameters v_a and ω_a are the actual robot velocities, accounting for lag. Error velocity states are defined, given by $v_a = e_v + v_d$ and $\omega_a = e_\omega + \omega_d$. These states are substituted into (6.1), forming,

$$f(x) = \begin{bmatrix} -(e_v + v_d)\cos(\alpha) + v_r \cos(\theta) \\ , \frac{(e_v+\omega_d)\sin(\alpha)}{e} - \frac{\omega_r \sin(\theta)}{e} - \omega_r \\ \frac{(e_v+v_d)\sin(\alpha)}{e} - \frac{\omega_r \sin(\theta)}{e} - e_w - w_d \end{bmatrix}. \tag{6.2}$$

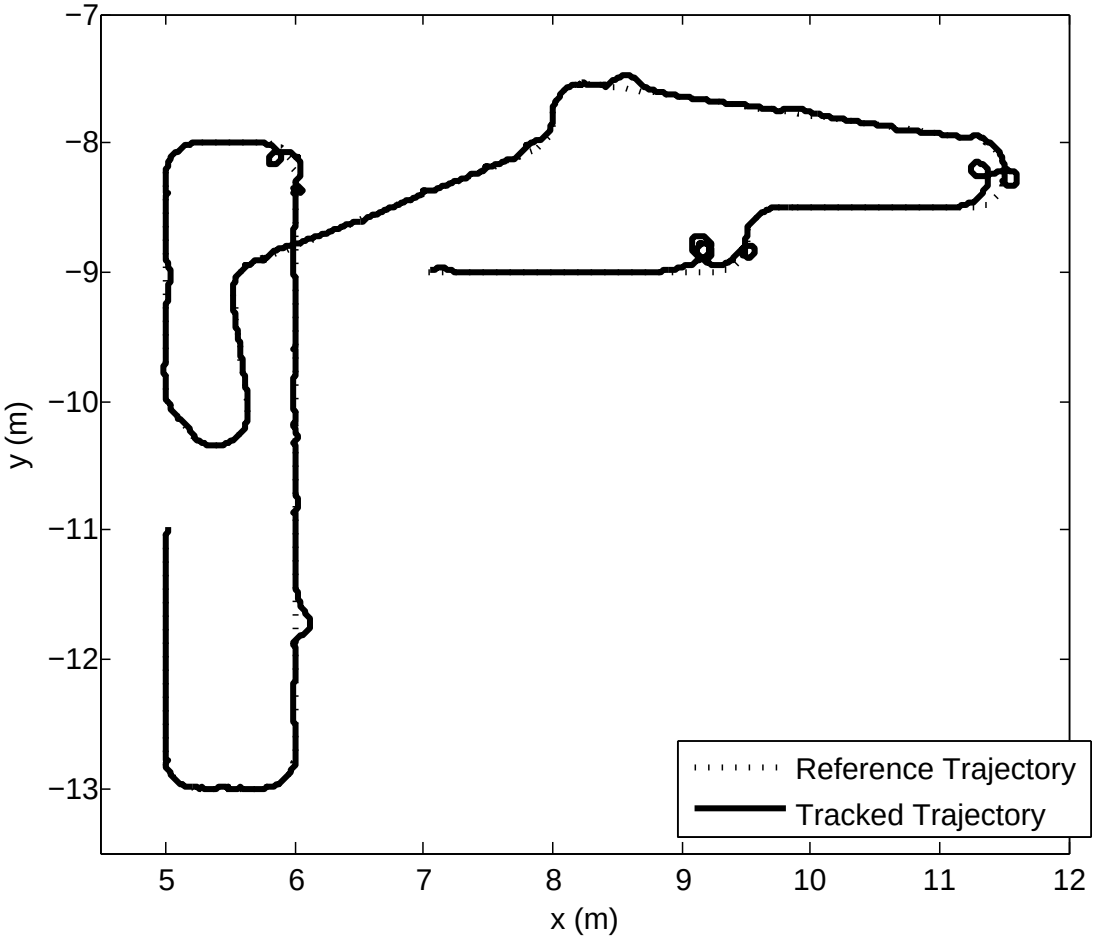

Figure 6.32. Simulation with controller output quantized at 30 Hz.

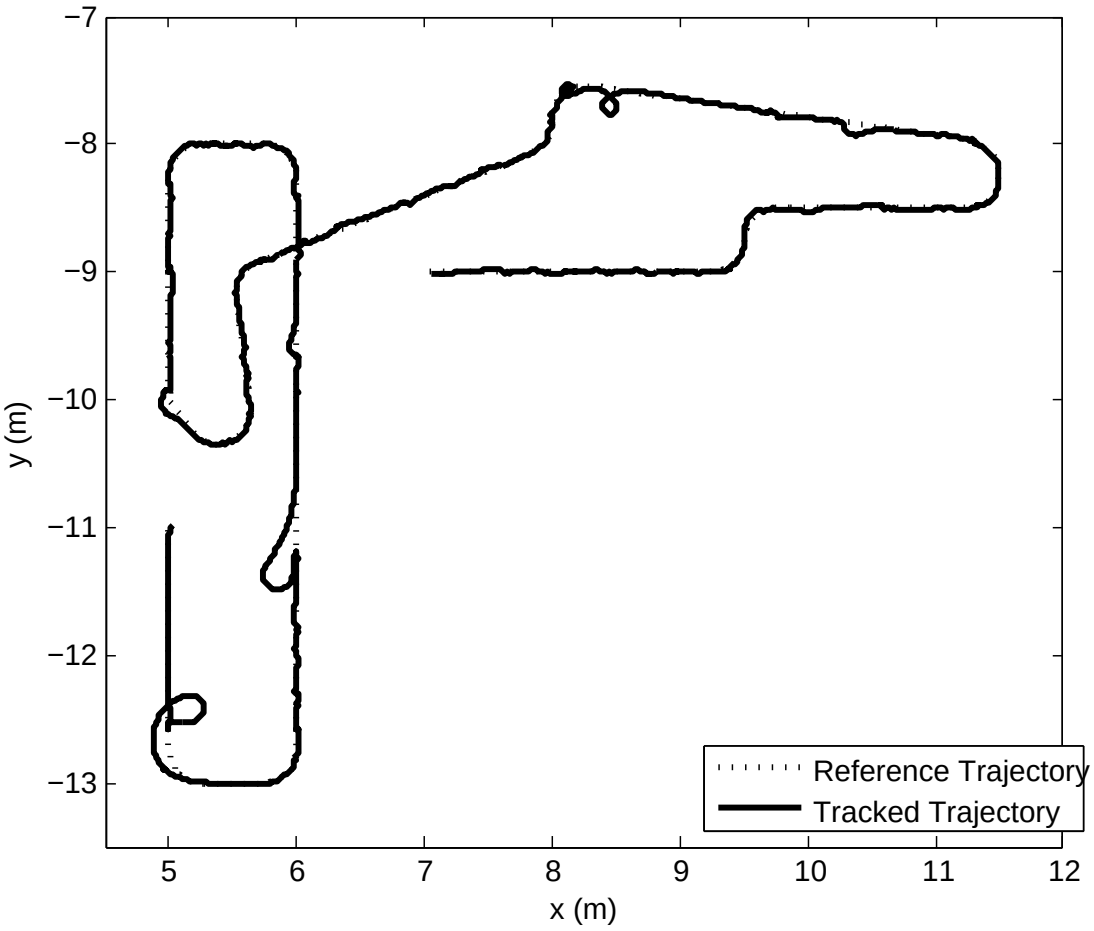

Figure 6.33. Simulation with controller output quantized at 20 Hz.

The nonlinear control laws (4.39) and (4.42) are substituted into (6.2), resulting in,

$$f(x) = -\Big(ev + \Big(-k1\ esqrt\left(1+\epsilon-\cos\left(2\,\theta\right)\right)\tanh\left(-e + rsqrt\left(2\right)sqrt\left(1+\epsilon-\cos\left(2\,\theta\right)\right)\right) + vr\,e\cos\left(\theta\right) \tag{6.3}$$

The system equation for the robot, modeled with a delay, is,

$$f_{robot}(x) = \begin{bmatrix} -\frac{e_v}{\tau_v} \\ -\frac{e_\omega}{\tau_{omega}} \end{bmatrix}, \tag{6.4}$$

where τ_v and τ_ω are the measured time constants of the robot for linear and rotational velocity, respectively. States are defined as $[e, \theta, \alpha, e_v, e_\omega]$.

The full nonlinear system becomes,

$$f_{sys}(x) = [-\Big(ev + \Big(-k1\ esqrt\left(1+\epsilon-\cos\left(2\,\theta\right)\right)\tanh\left(-e + rsqrt\left(2\right)sqrt\left(1+\epsilon-\cos\left(2\,\theta\right)\right)\right) + vr\,e\cos \tag{6.5}$$

The equilibrium point is chosen as,

$$[e, \theta, \alpha, v, e_v, e_\omega] = [\epsilon, 0, 0, v_r, 0, 0]\,. \tag{6.6}$$

The nonlinear system is linearized by calculating the Jacobian about this equilibrium point. For space reasons, the Jacobian is not shown.

The state transition matrix of the linearized system is calculated by a fourth order approximation, given by,

$$\phi_{NL} = I + A * T + A^2T^2/2! + A^3T^3/3! + A^4T^4/4!. \tag{6.7}$$

The system is then discretized by transforming it into the z domain by,

$$F(z) = z * I - \phi_{NL}. \tag{6.8}$$

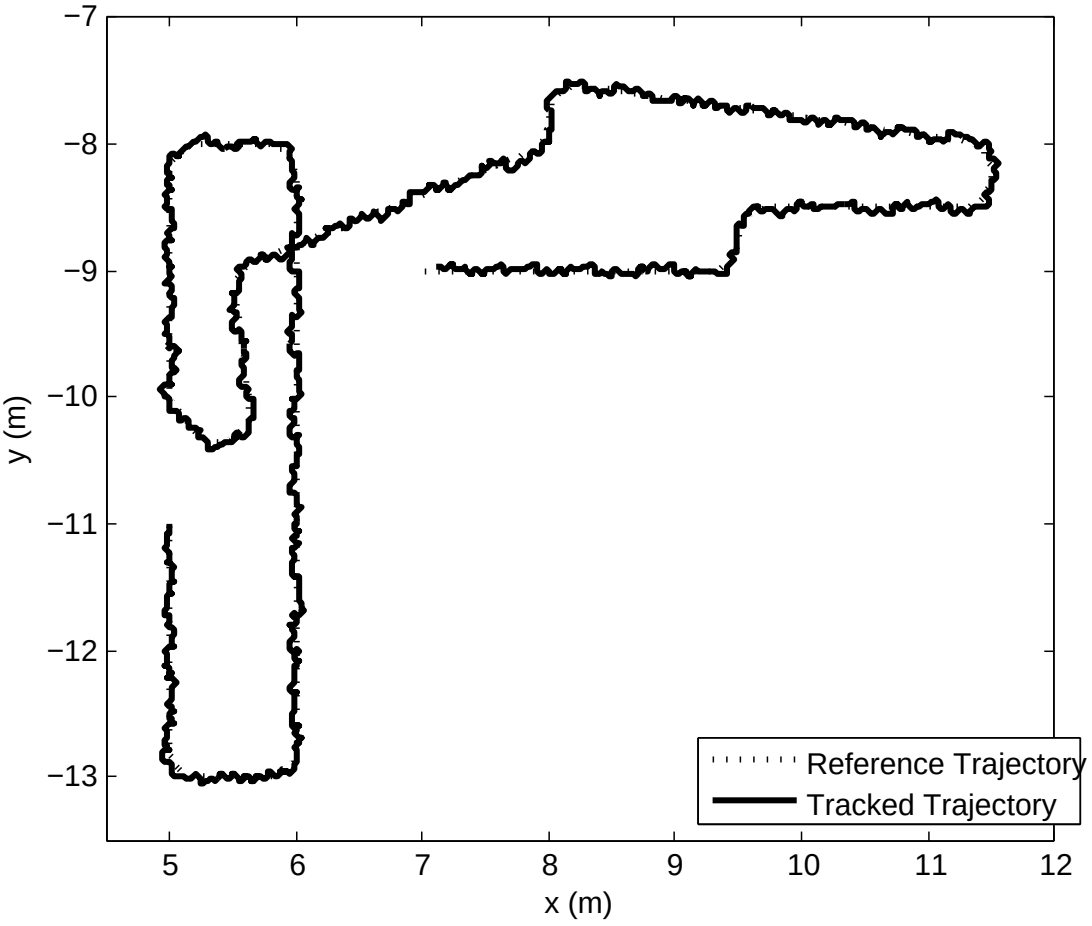

Figure 6.34. Simulation with controller output quantized at 10 Hz.

The desired system parameters are then substituted in to discrete linearized system: A sampling frequency of $T = 1/30$, dynamic extension gains of $k_v = 3.0$, $k_c = 3.0$, controller gains of $k_1 = 0.85$, $k_2 = 0.5$, controller parameters of $\epsilon = 0.03$, $r = 0.2$, velocities of $v_r = 0.1$, $v = v_r$, and robot time constants of $\tau_v = 0.5$, $\tau_\omega = 0.5$. The desired root magnitude is calculated by $10^{(T log(\tau))}$, where $T = 1/30$ seconds, and the desired time constant $\tau = 0.3$ seconds. This results in a root magnitude of $z = 0.9117327878$.

The discrete system is evaluated with $k_1 = 3.0$, $k_2 = 5.0$, and $k_v = 3.0$, $k_c = 3.0$. The parameter ϵ and path manifold radius r are varied to evaluate their effect on stability. The sampling frequency is 30 Hz, and the reference velocity is 1.0 meters be second. The rotational reference velocity, ω is equal to zero. Figure 6.35 shows the z transform root magnitudes of three of the system states under varying r and ϵ. The root magnitude is close to the desired value of 0.91. In Figure 6.36, the root magnitude of the remaining two states is shown. The value is generally lower than desired, but stable. Changing the parameter ϵ has no effect on the roots, but smaller values of r cause the magnitude to drop. Sufficiently small values of ϵ cause the root magnitude to increase.

The damping ratios of the discrete system for all five states is given in Figure 6.37. These states are critically damped, and varying the parameters r and ϵ has no effect. The parameter r must increase significantly in order to achieve more damping. At the lowest point around $r = 0.1$ meters, the damping ratio is sufficient for the desired system performance. As with the z transform root magnitudes, the parameter ϵ has no effect on damping.

The discretized system is evaluated for stability at different reference velocities, using the same parameters as above, with the exception of $k_v = 15$ and $k_c = 45$. Figure 6.38 shows a plot of the z transform roots for velocities between zero and two meters per second. Three of the roots remain stable for all velocities, but the last two roots become unstable at $v_r = 1.6$ meters per second. The magnitude of all roots must be less than one for the system to meet stability criteria. The damping ratios are given in Figure 6.39.

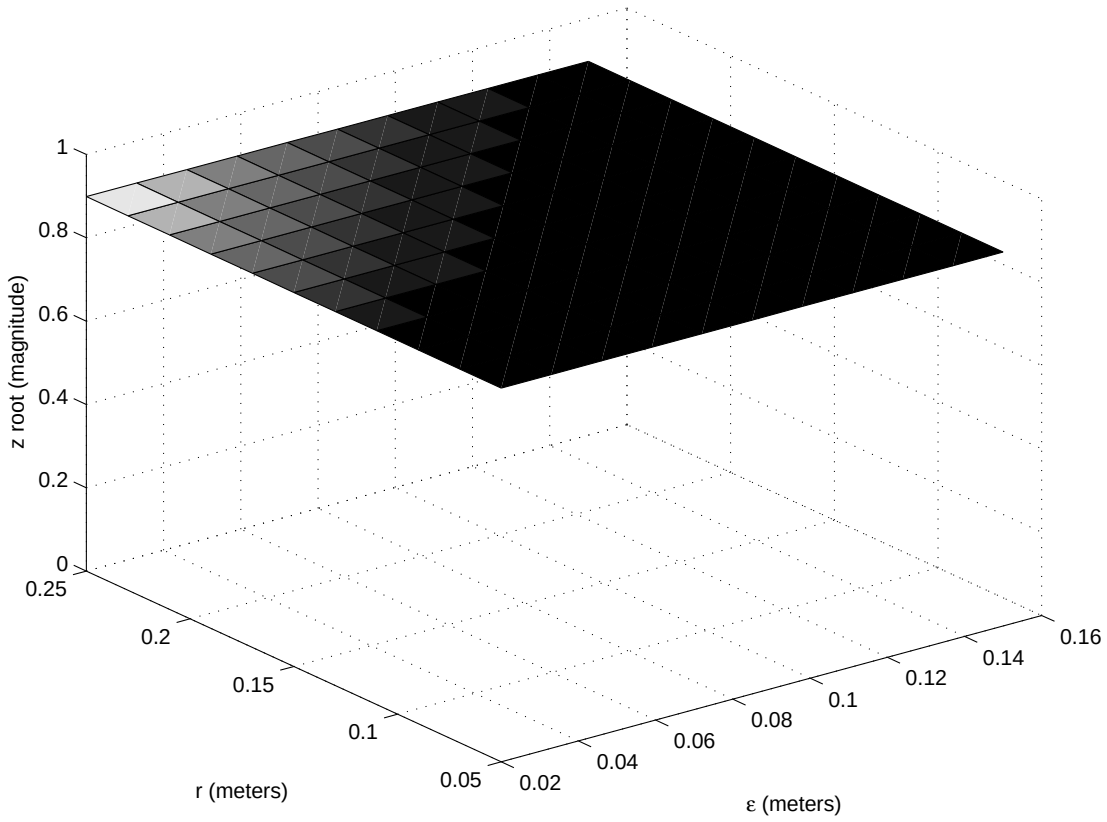

Figure 6.35. Z transform root magnitude of discrete system, varying r, ϵ, states 1,2,3.

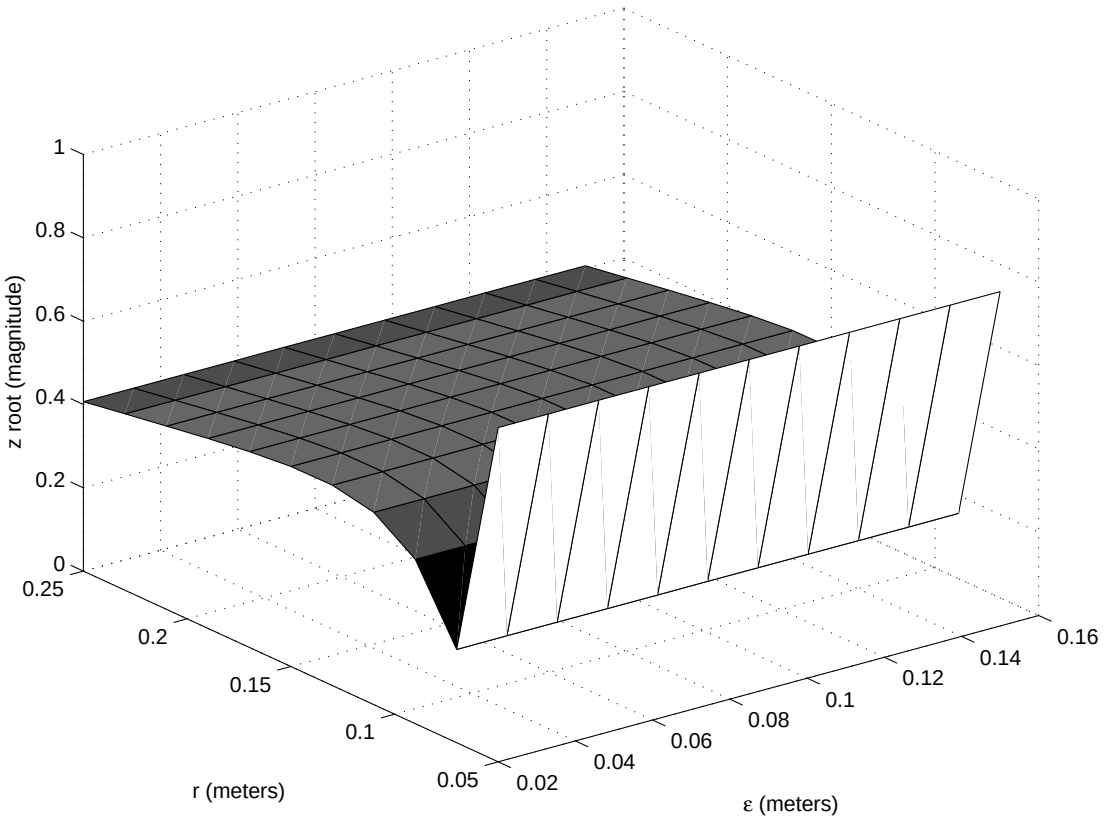

Figure 6.36. Z transform root magnitude of discrete system, varying r, ϵ, states 4,5.

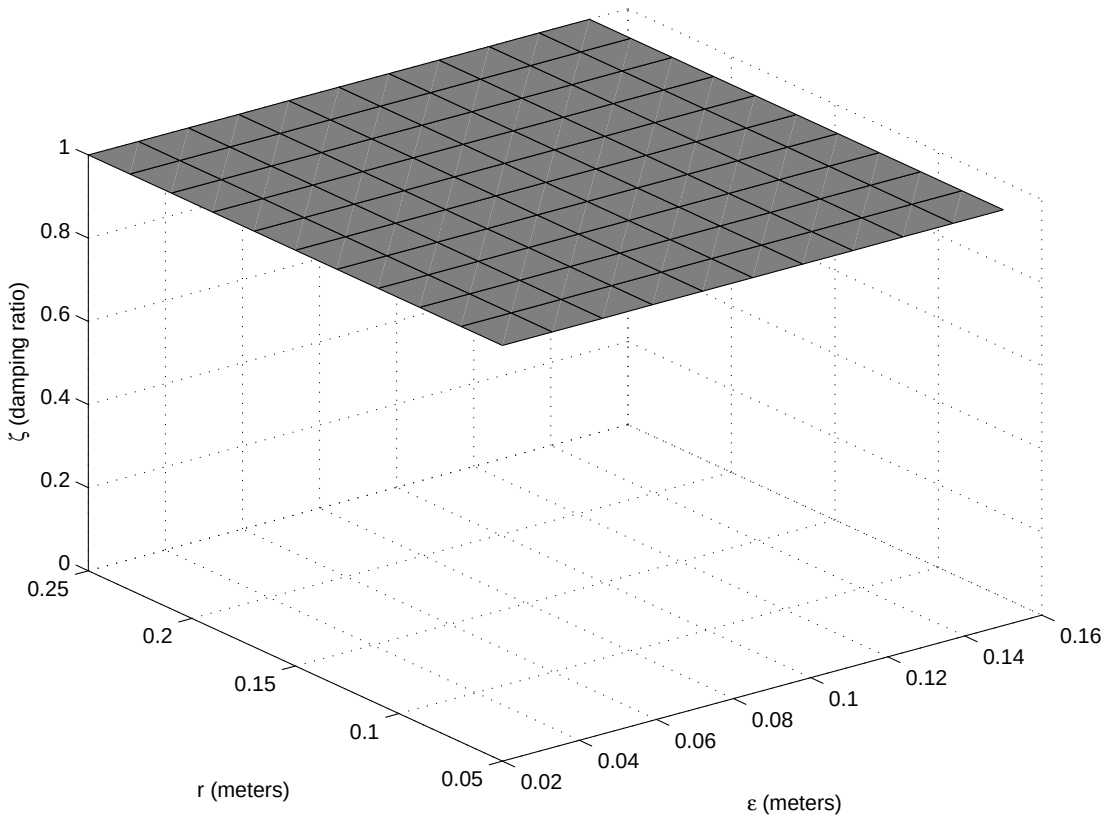

Figure 6.37. Damping ratios of discrete system, varying r, ϵ, all states.

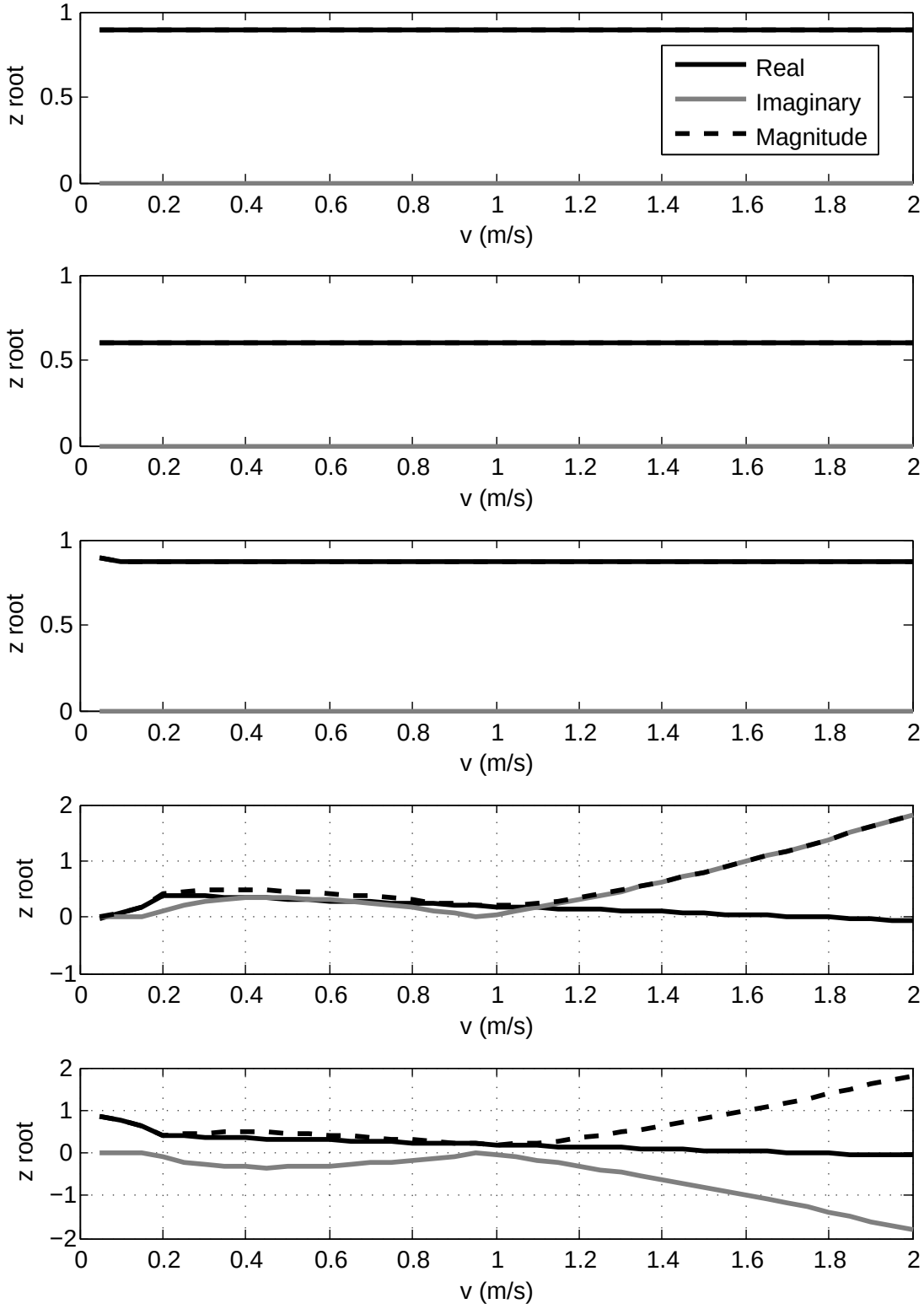

Figure 6.38. Z transform roots with varying reference velocity.

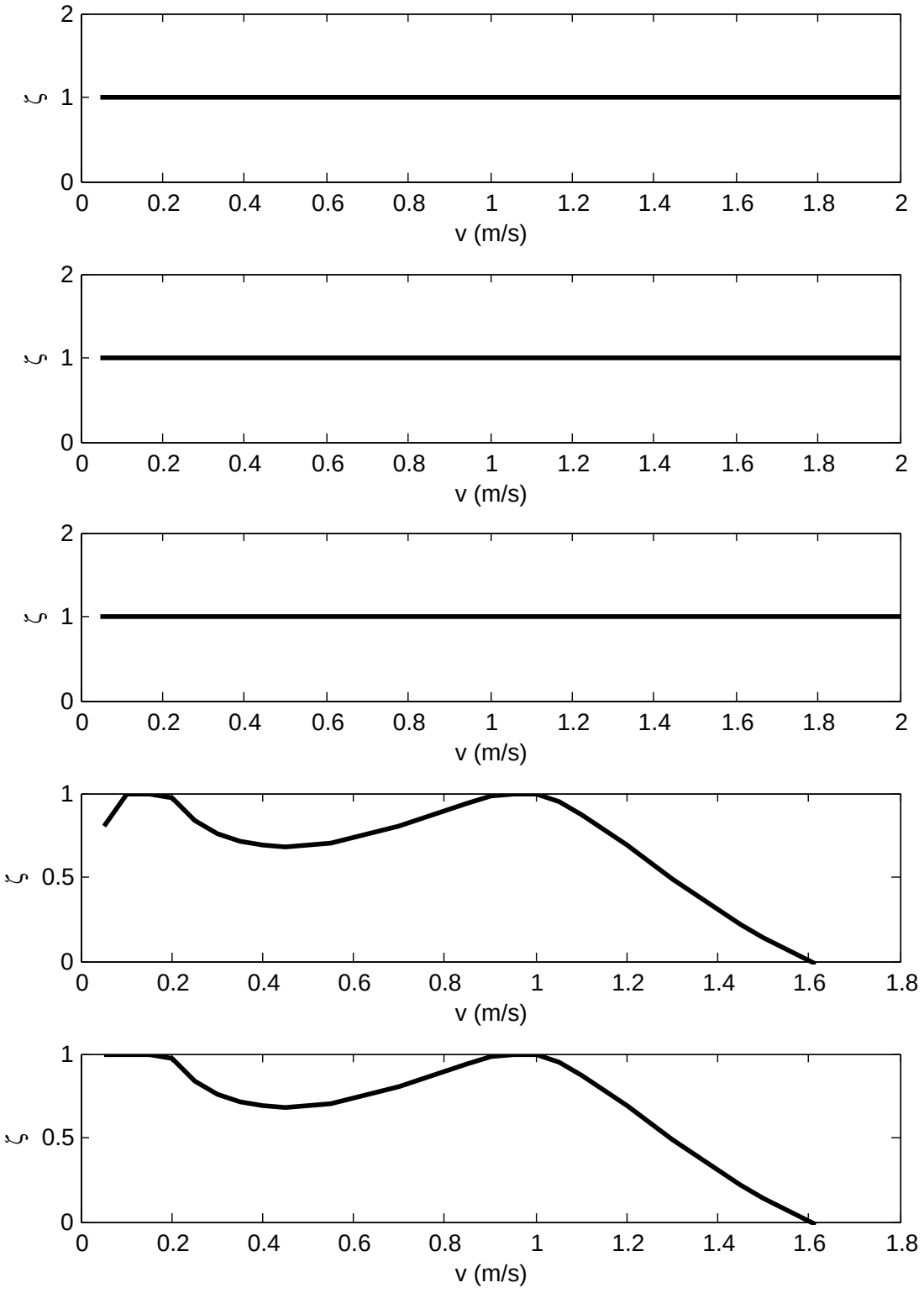

Figure 6.39. Z transform damping ratios with varying reference velocity.

6.3.5 Simulation Results

The simulation results for the parameter tuned controller are presented in this subsection. The reference trajectories range from a straight line segment, to paths with multiple curves. The performance of the controller is evaluated in preparation for its implementation in Emulab Mobile.

For a baseline simulation test, a trajectory without any curves is created. The kinematic controller, implemented in simulation with SIMULINK, tracks the reference trajectory closely. Figure 6.40 gives the resulting trajectory compared to the reference trajectory. Initial conditions for this simulation are $e(0) = 0.0283$, $x(0) = 4.9800$, $y(0) = -10.9800$, and $\phi(0) = -\frac{\pi}{4}$. These initial conditions are representative of expected initial conditions for robots in Emulab Mobile.

A linear velocity profile is used, as shown in Figure 6.41. A maximum velocity of 0.1m/s is chosen, which would be a reasonably slow speed for robots on the testbed. The controller in this simulation is run at 1000 Hz, which is much faster than it will be run in implementation. The total simulation time resulting from the chosen maximum velocity is 10 seconds.

The response of the system given in (4.3) is presented in Figure 6.42. The initial polar distance error, e, is slightly less than 3cm, but climbs to a peak of 4.5cm before converging below 2cm. The convergence rate could be improved by adjusting the controller and dynamic extension gains discussed in Section 4.4. As seen in Figure 6.40, the controller tracks the path well, but lags behind the reference trajectory. The lateral position error is minimal in this case, as evidenced by the rapidly decreasing values of the θ and α states after two seconds simulation time in Figure 6.42.

Velocity output commands from the simulated controller are shown in Figure 6.43. v originally does not match the reference velocity profile in Figure 6.41, but settles after two seconds. Behavior of the ω velocity command is similar, as the controller compensates for the nonzero initial states.

The reference and simulated trajectories are given in Figure 6.44. The velocity profiles are presented in Figure 6.45 and Figure 6.46.

The response of the system given in (4.3) is presented in Figure 6.47. The disturbance from the discontinuity in curvature is apparent in the jumps in all three states at 0 seconds.

The velocity output commands are shown in Figure 6.48. The controller must compensate for the curvature discontinuity at the boundary points of the arc.

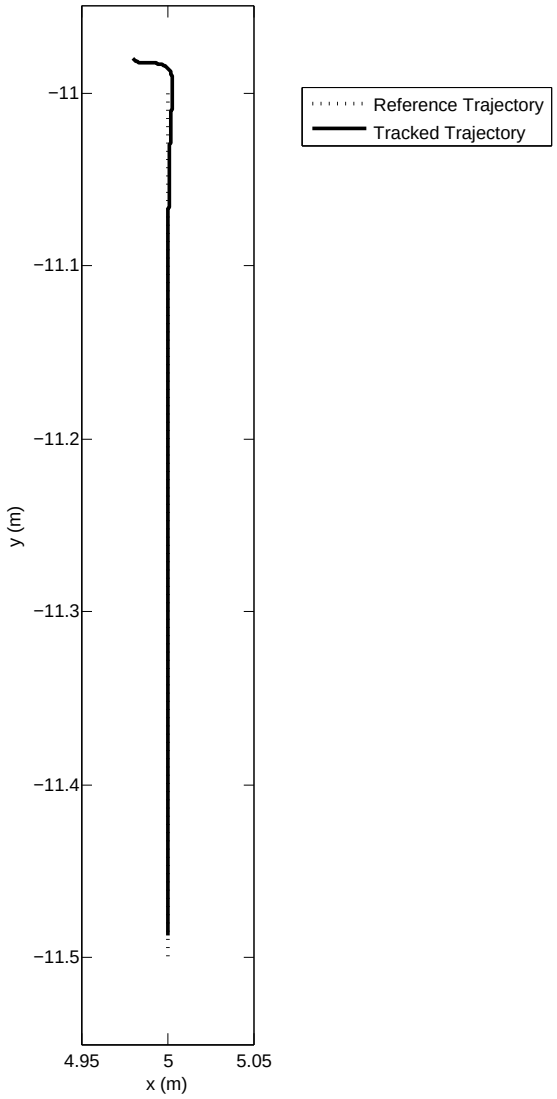

Figure 6.40. Straight line trajectory: Simulated trajectory.

6.3.6 Simulation of Trajectory Tracking Controller Functions in RMCD

Before integration into RMCD, the trajectory tracking functions are tested and compared to results from simulation. The major system component outputs tested include gain calculation, the Cartesian to Polar state transformation functions, the main control law, and the dynamic extension. A test suite for the RMCD motion control functions is created, and a framework built in MATLAB to evaluate the results and compare them to simulation results.

A second curved path is tested in simulation, as shown in Figure 6.49. The Cartesian states output from this simulation are given as input to the functions implemented in

RMCD, as discussed in Subsection 7.4.1. The parameters used in this simulation are presented in Table 6.1.

The implementation of the Polar state transformation (4.3) in RMCD is compared to the Polar states calculated in the SIMULINK simulation of the trajectory tracking controller. The states calculated by RMCD are identical to the simulated signals, with the exception of $\dot{\theta}$. This is caused by differences in numerical differentiation algorithms between the two applications.

The simulated controller velocity outputs v and ω, and their derivatives $\dot{v}$ and $\dot{\omega}$ are plotted in Figure 6.51. The signals match, though the noise in the derivatives alters the ω signal significantly.

The output of the dynamic extension in simulation is compared to the output of the dynamic extension in RMCD in Figure 6.52. The signal v from RMCD matches the simulated signal, except for discontinuities between zero and five seconds.

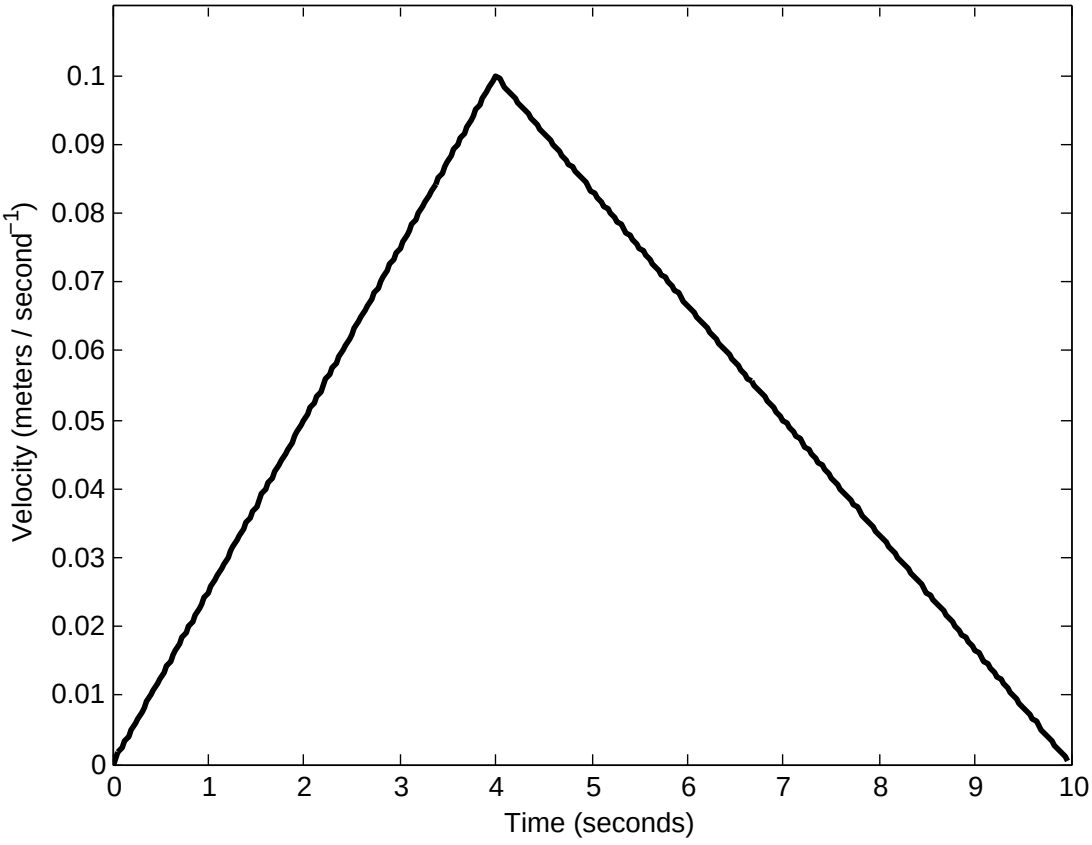

Figure 6.41. Reference velocity profile for trajectory in Figure 6.40

r	0.2 meters
ϵ	0.03 meters
k_1	3.0
k_2	5.0
k_v	3.0
k_c	3.0

Table 6.1. Controller parameters used in RMCD function to simulation comparison test.

6.3.7 Filtering of Derivatives in RMCD Controller Implementation

The numerical differentiation method employed to obtain $\dot{v}$, $\dot{\omega}$, and $\dot{\theta}$ has significant noise, and must be filtered to preserve the stability and boundedness of the motion controller. Digital filtering is used for these signals to attenuate high frequency noise.

An example trajectory with a single curve is presented in Figure 6.53. The tracked trajectory is the result of a SIMULINK simulation of the motion controller and kinematics, as discussed in Section As outlined in Subsection 6.3.6, the simulated data is passed to the motion controller functions implemented as part of RMCD. (RMCD is discussed in detail in Subsection 2.5.2.) The corresponding velocity profile is given in Figure 6.54. A maximum velocity of 0.1 meters per second is chosen, with linear velocity ramps at the trajectory boundaries.

Figure 6.55 shows the resulting system states as calculated by RMCD. Without filtering, the first differential of θ is noisy, with saturation in the region of the single curve in the path. The condition of this signal is detrimental to the performance of the main controller, as the control law (4.42) for ω includes $d\theta$.

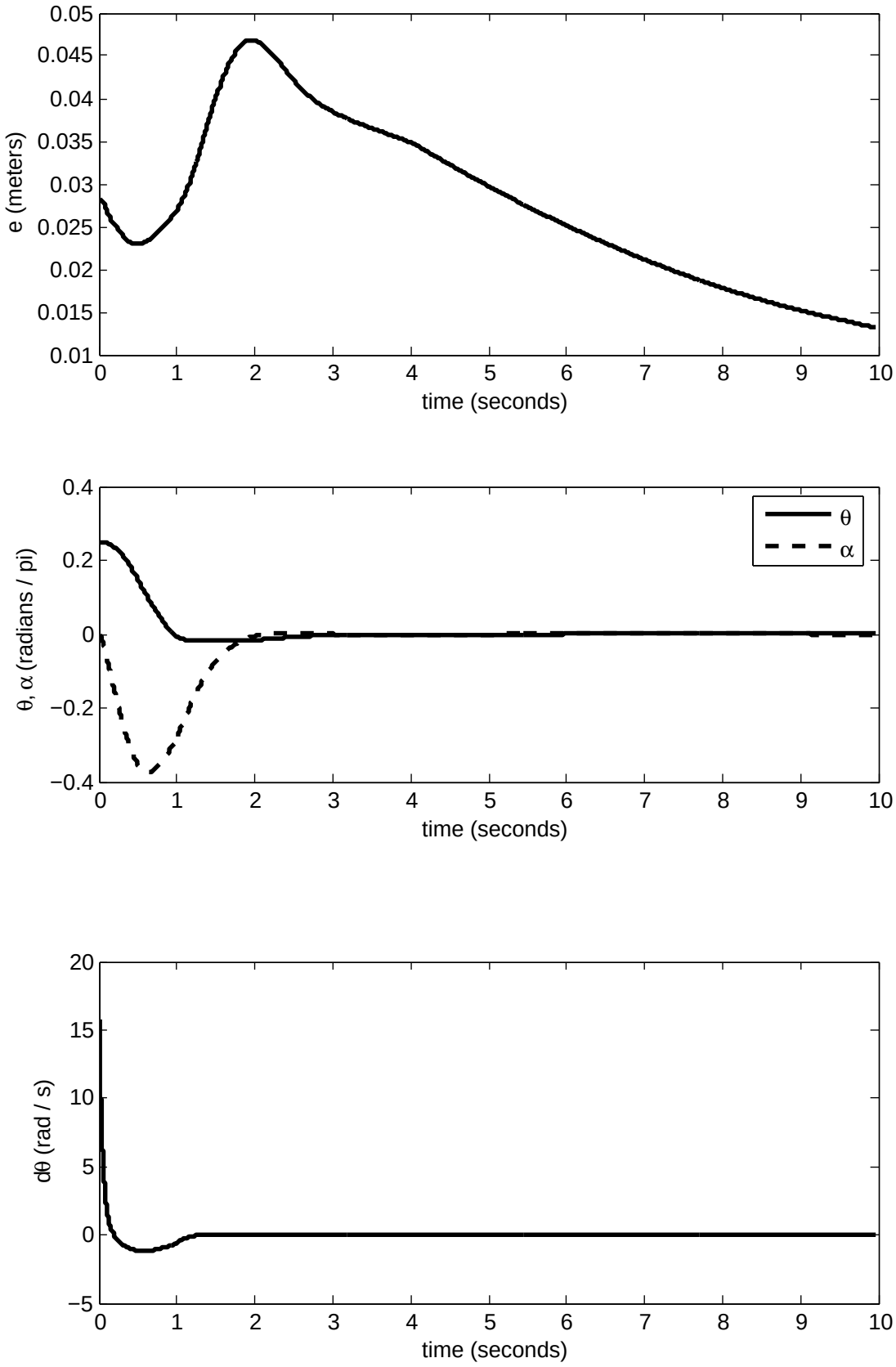

Figure 6.42. System response of simulation given in Figure 6.40.

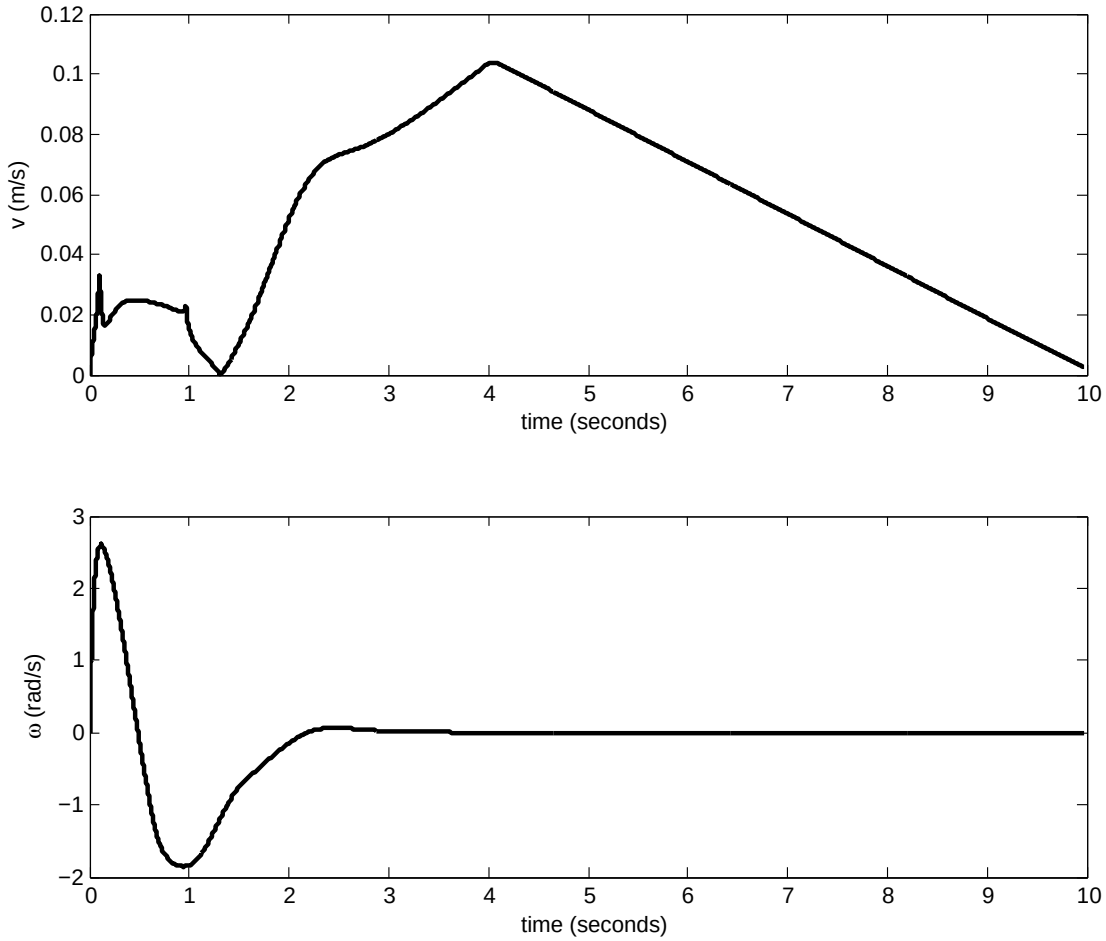

Figure 6.43. Simulated controller velocity commands for trajectory given in Figure 6.40.

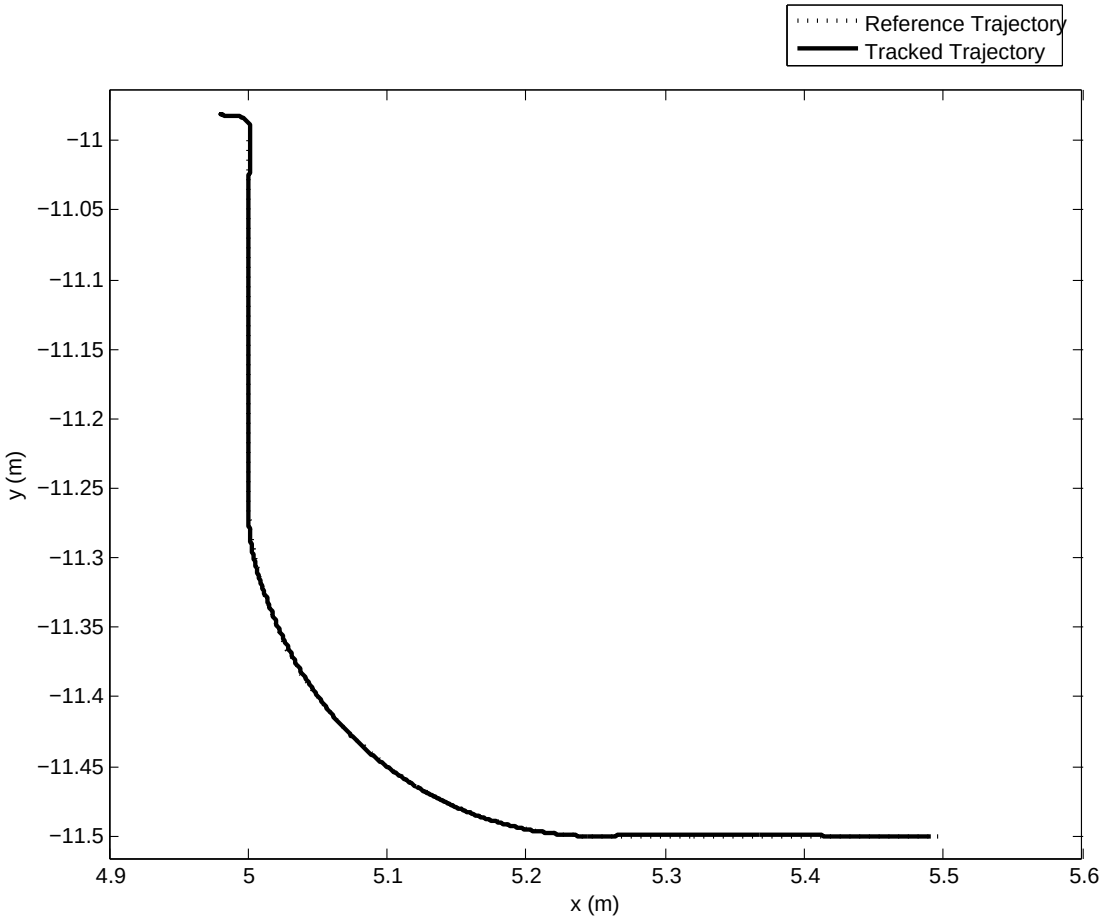

Figure 6.44. Simulated trajectory with a single curve.

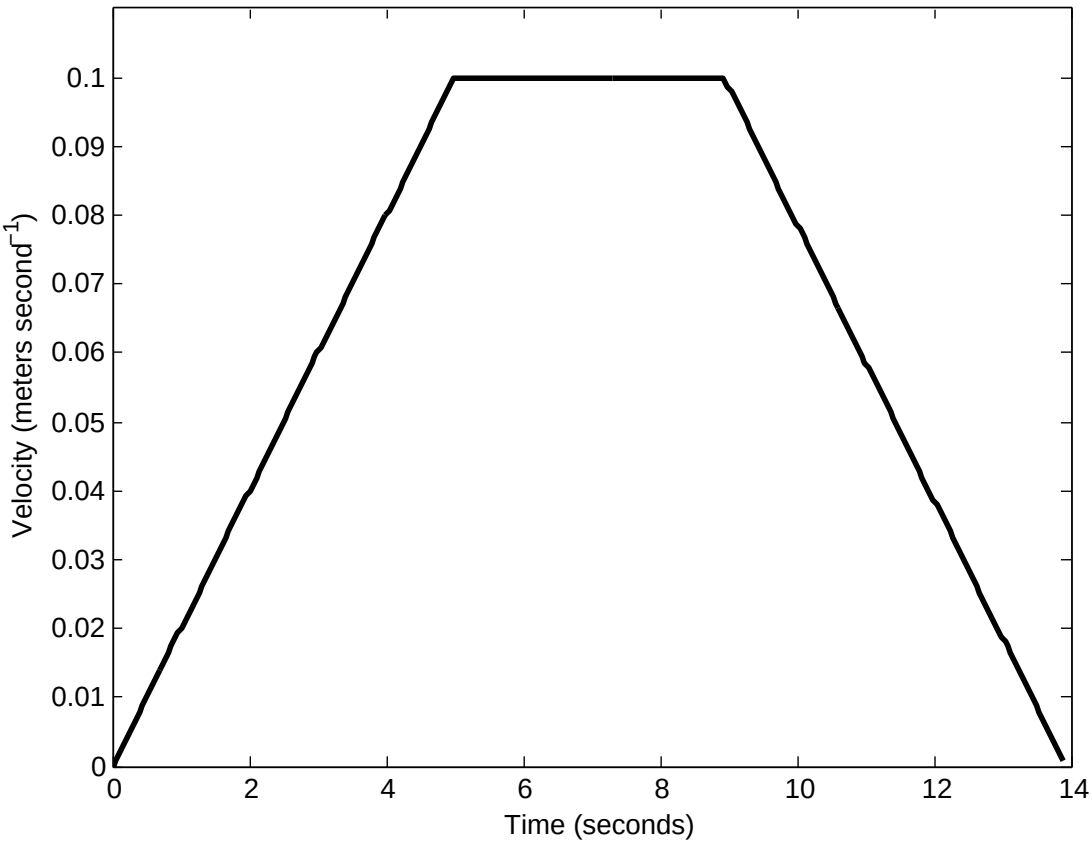

Figure 6.45. Reference velocity profile for trajectory in Figure 6.44

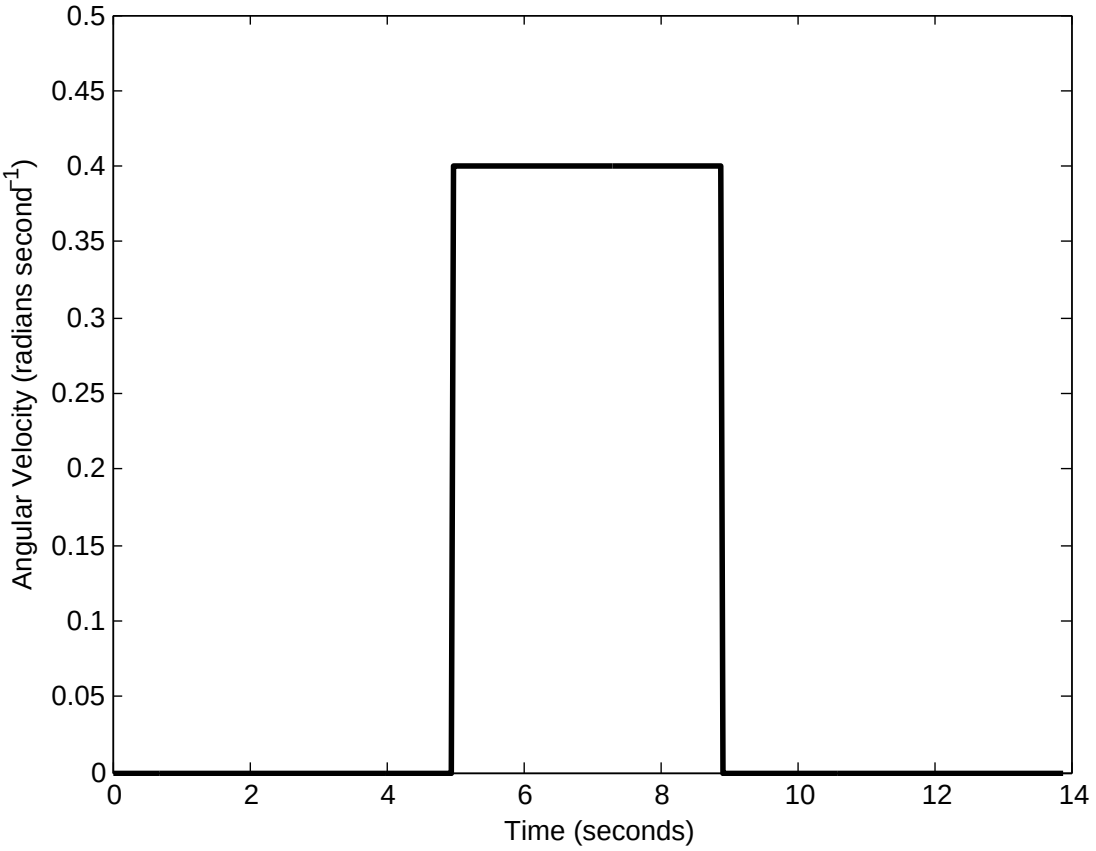

Figure 6.46. Reference angular velocity profile for trajectory in Figure 6.44

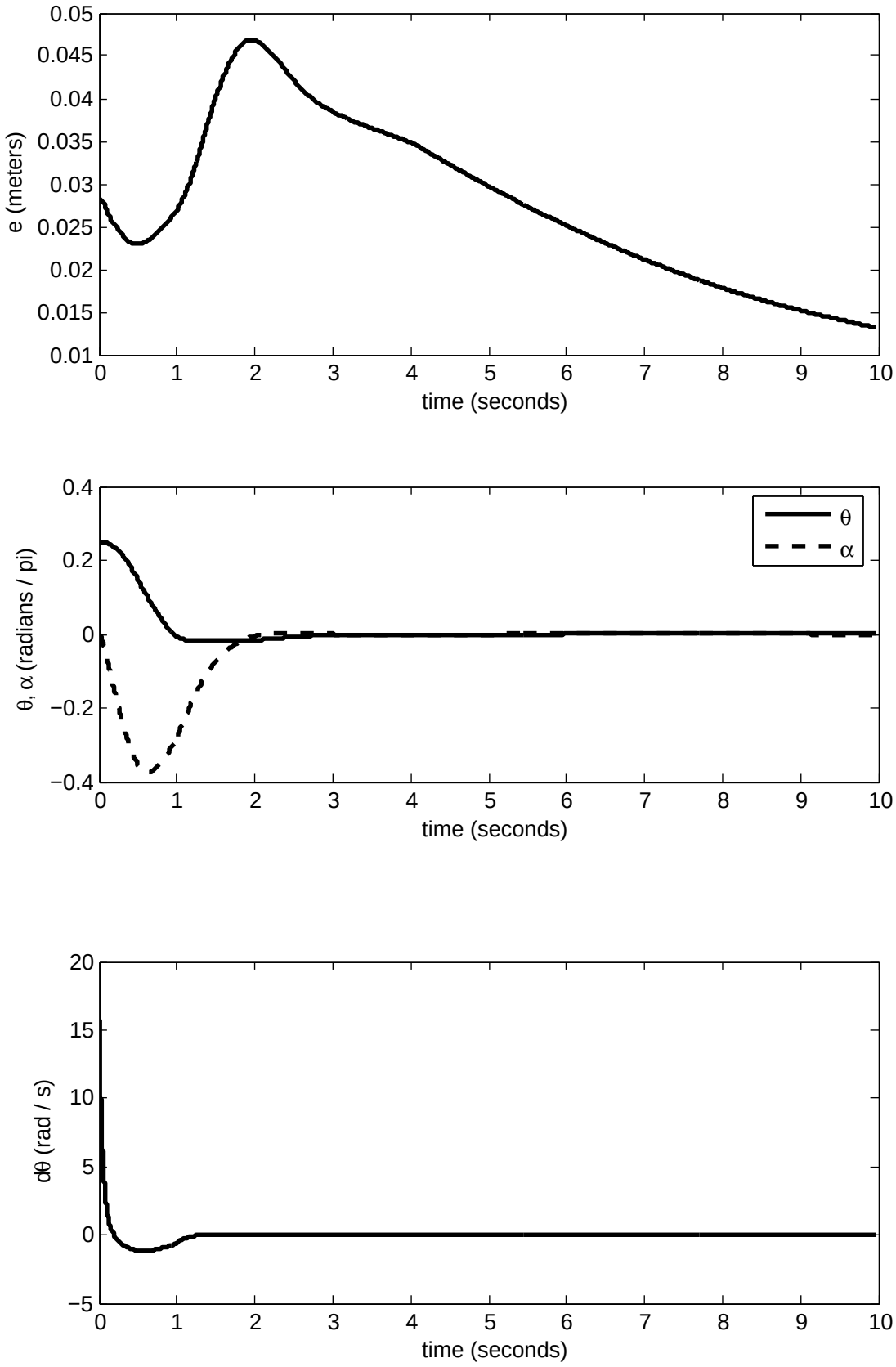

Figure 6.47. System response of simulation given in Figure 6.44.

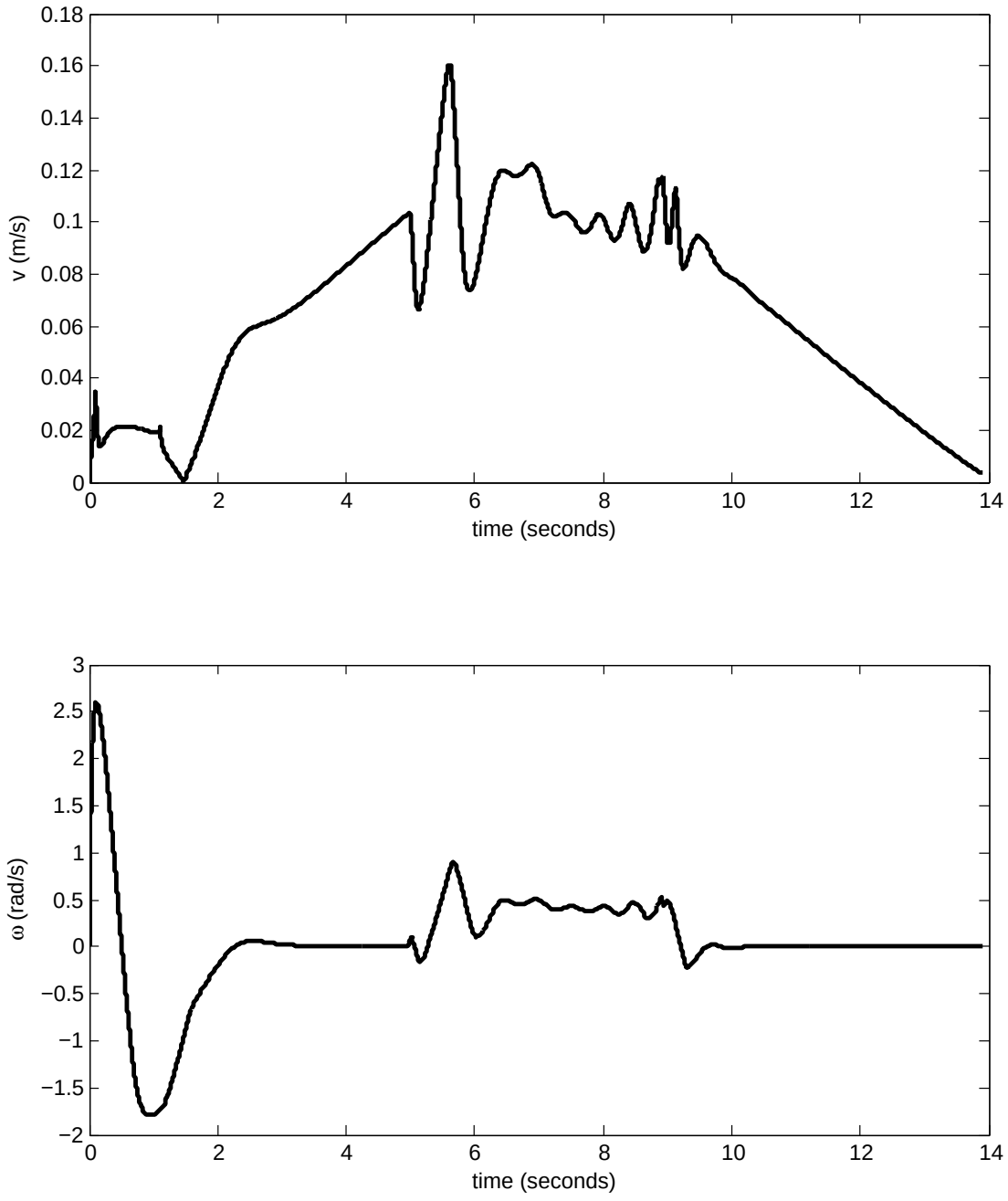

Figure 6.48. Simulated controller velocity commands for trajectory given in Figure 6.44.

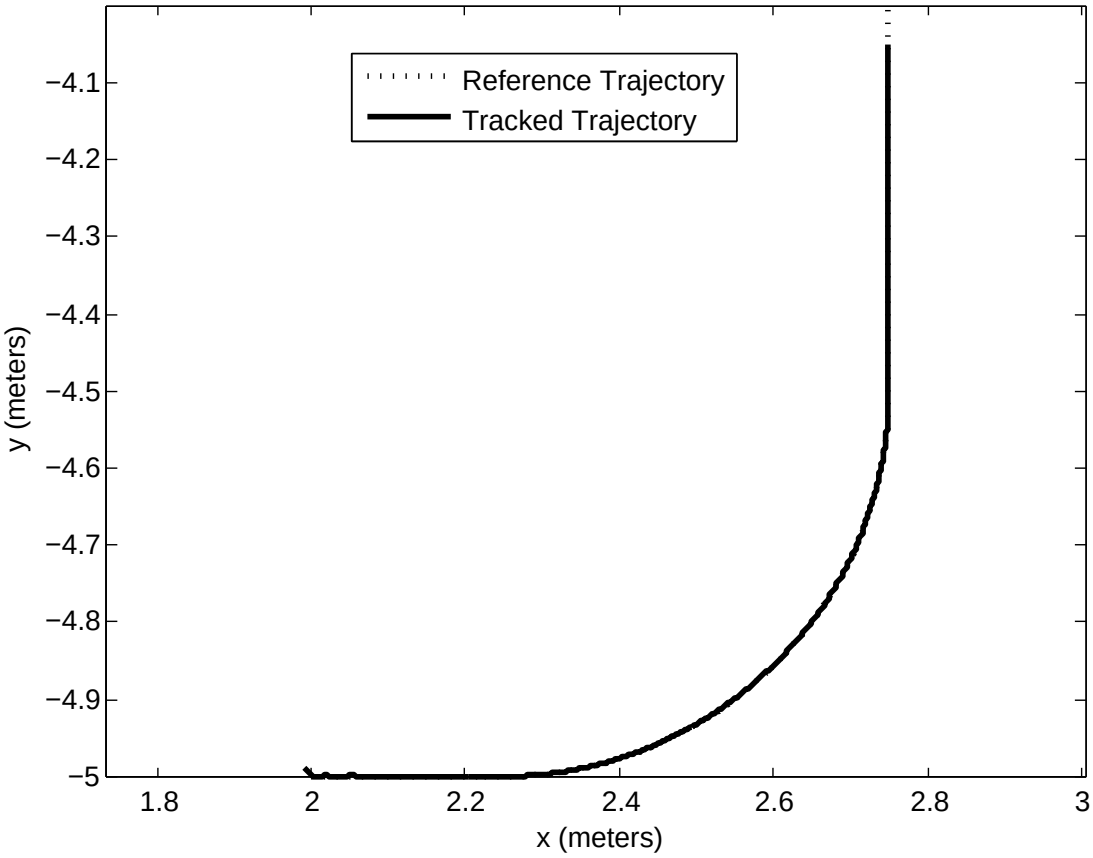

Figure 6.49. Simulated trajectory compared to RMCD functions for a curved path.

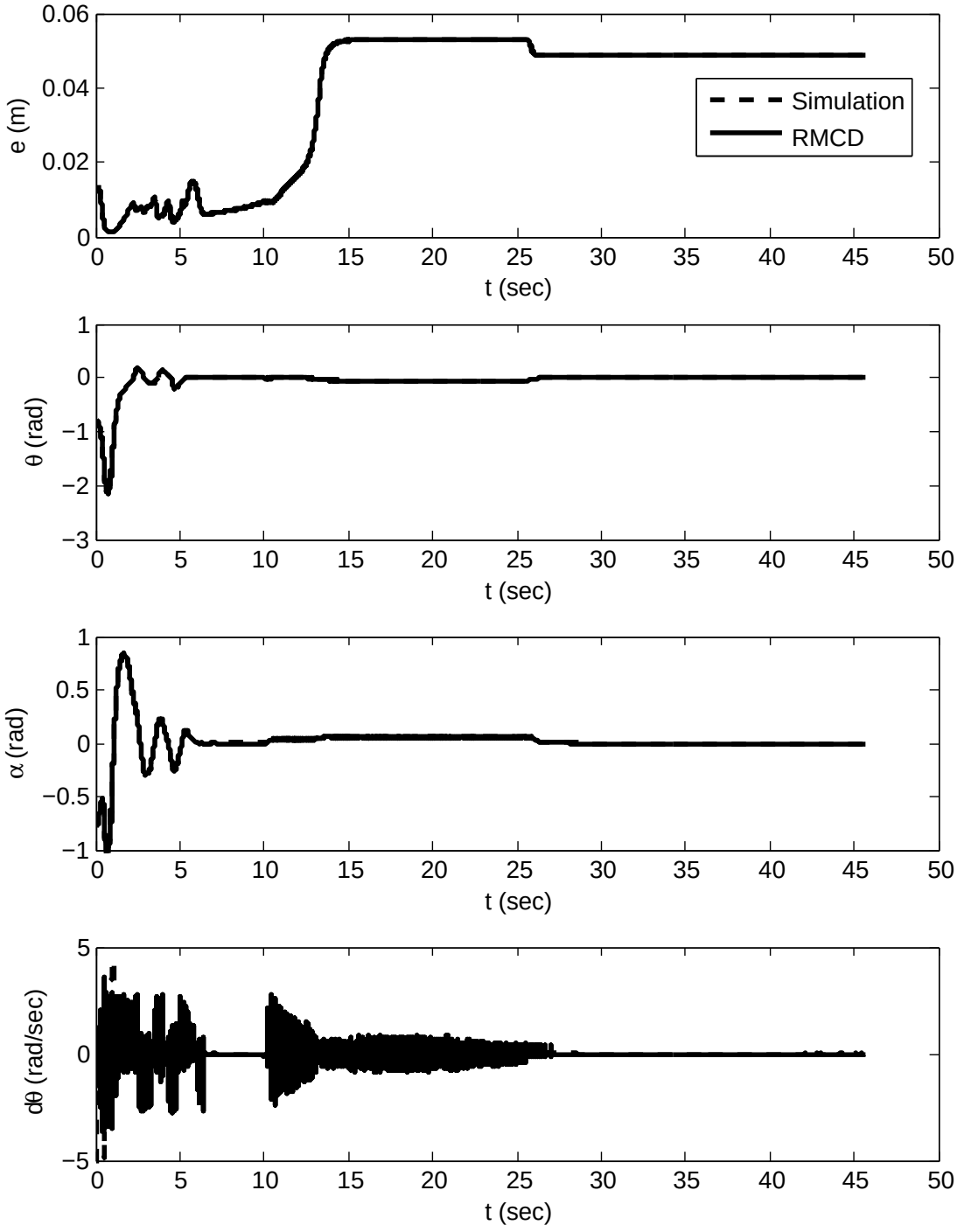

Figure 6.50. Simulated Polar state data compared to RMCD state data for a curved path.

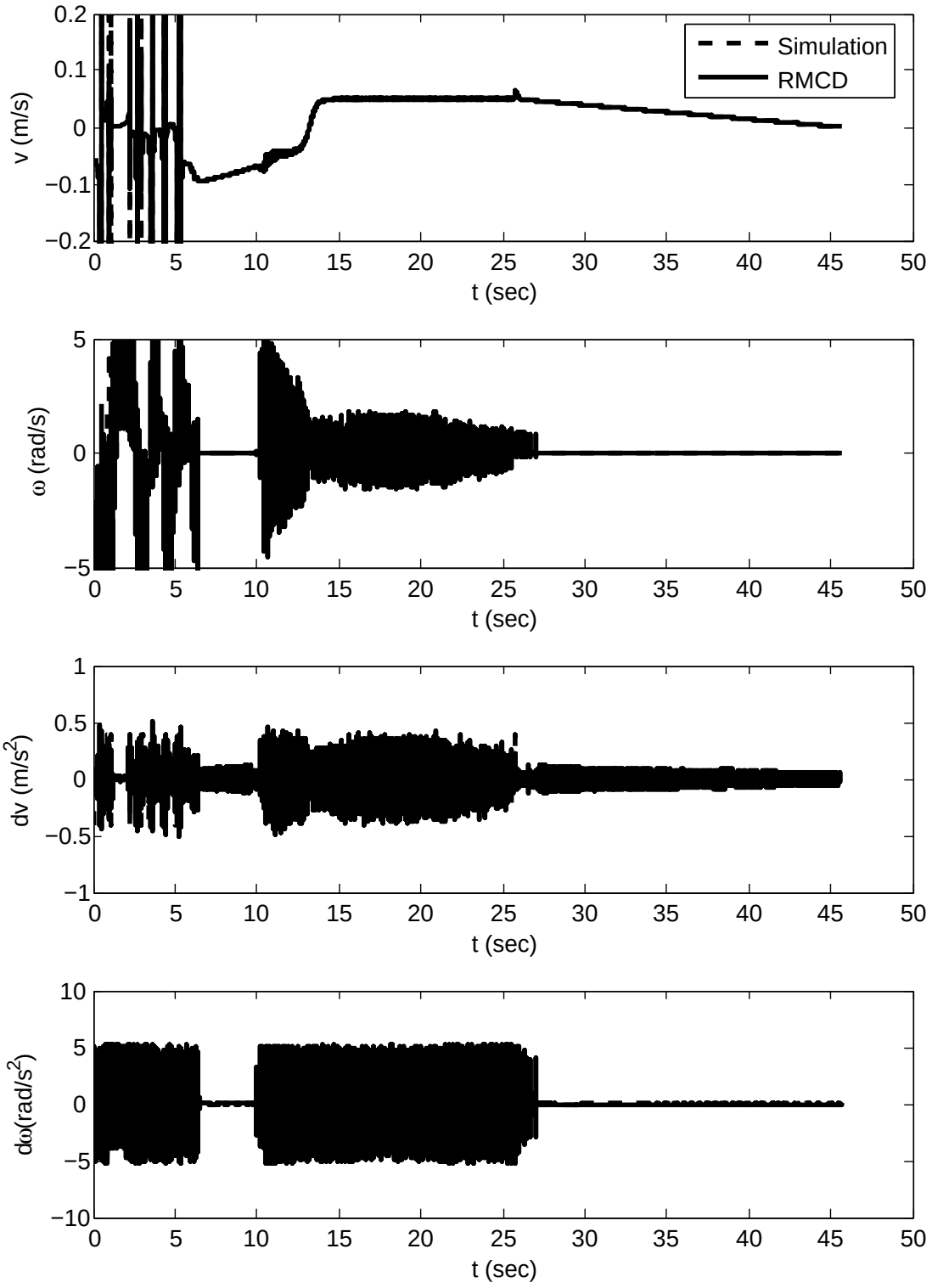

Figure 6.51. Simulated controller output compared to RMCD controller output for a curved path.

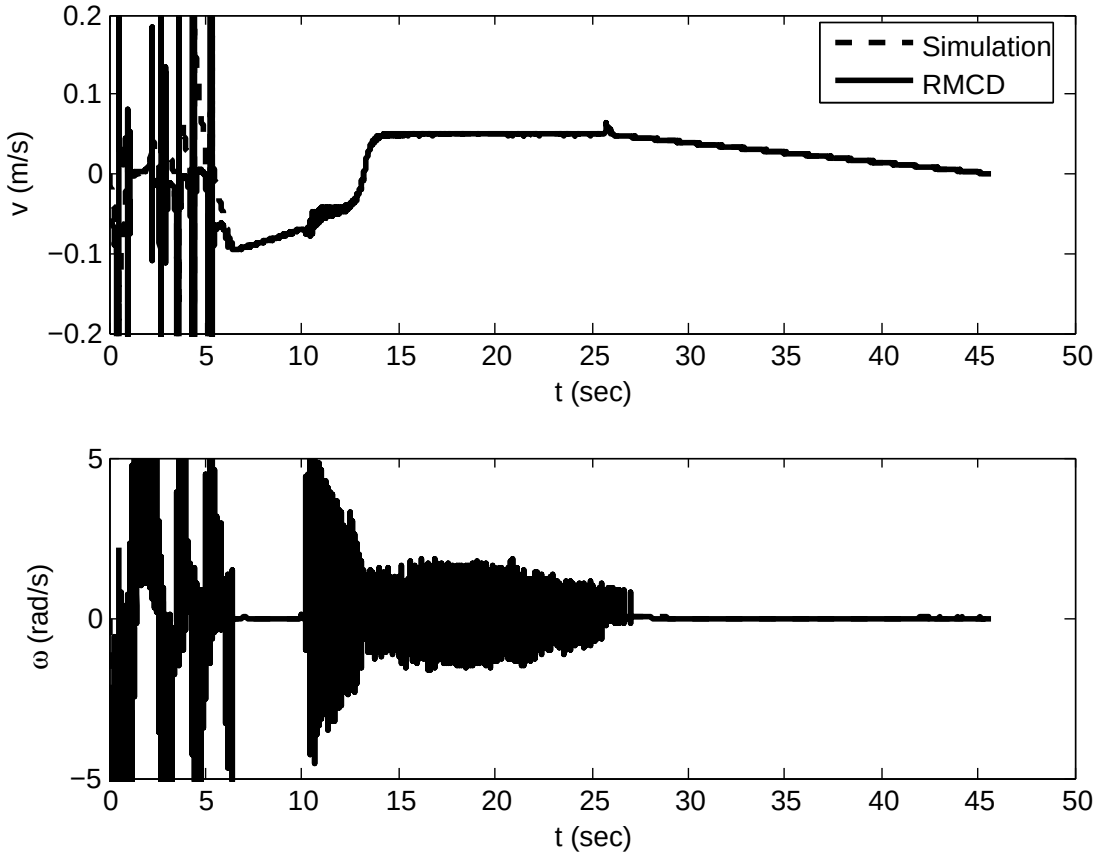

Figure 6.52. Simulation velocity output compared to RMCD velocity output for a curved path.

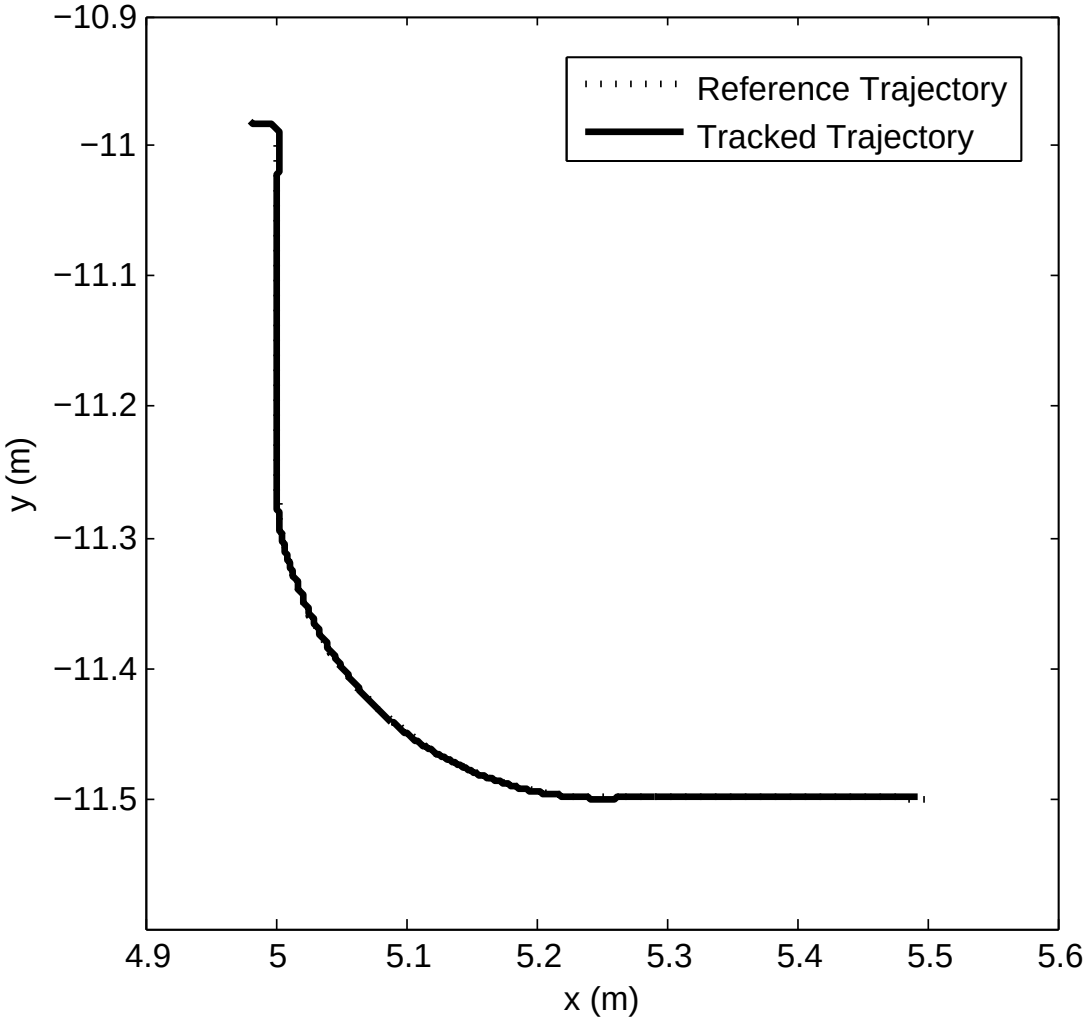

Figure 6.53. RMCD controller implementation without filtering, reference path.

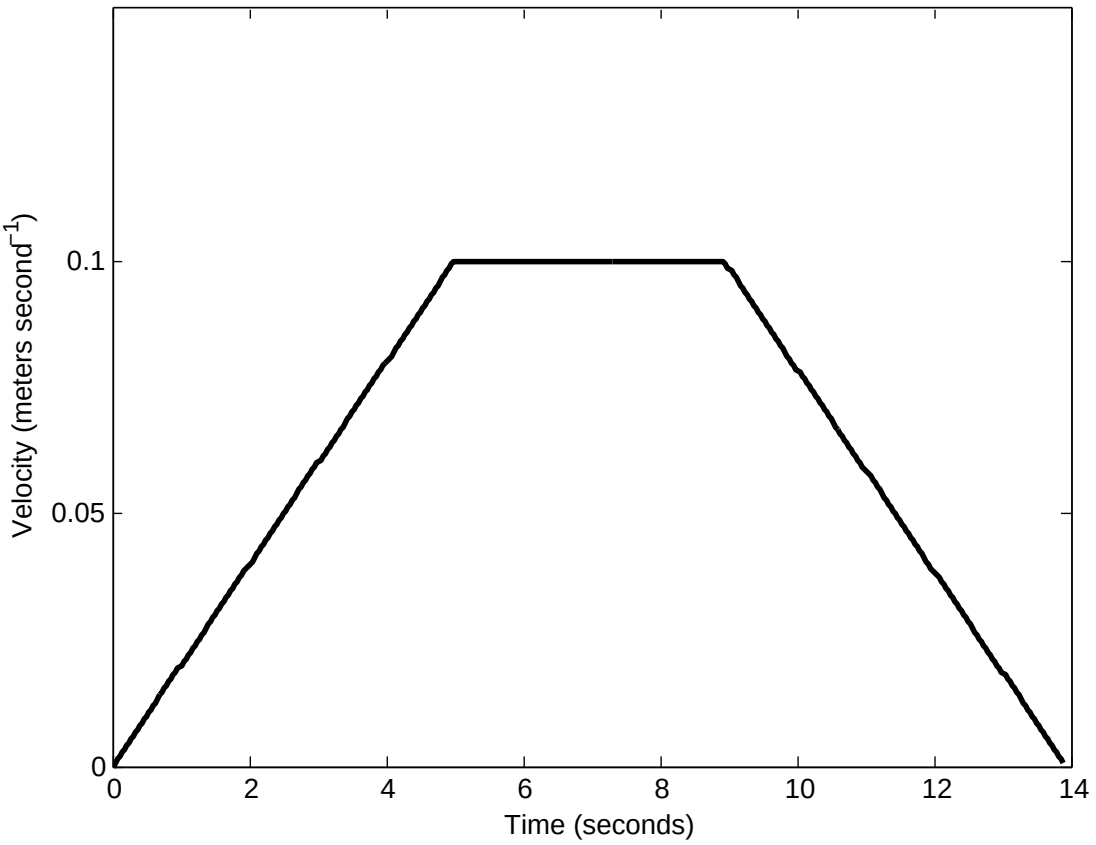

Figure 6.54. RMCD controller implementation without filtering, reference velocity profile.

The controller output is shown in Figure 6.56. The signal ω is noisy, just as the $d\theta$ signal. The first derivatives of the controller are also noisy, due to the absence of filtering. The signals saturate in this example, which keeps the motion controller output bounded.

The final output of the motion controller comes from the dynamic extension, which is discussed in Subsection 4.4.2. Figure 6.57 shows the output of this component, without any filtering of the output of the controller output derivatives. The dynamic extension in RMCD has significant noise and error compared to the dynamic extension as simulated using SIMULINK.

6.3.8 Run Time of Motion Controller in RMCD

It is important that the motion controller as implemented in RMCD, is capable of returning a set of wheel speeds in a sufficient amount of time. As position updates from the localization system come in at 30 Hz, the motion controller must complete all calculations within 0.03333 seconds, before another position update arrives. It is desired that compution for the motion control be as fast as possible, to reduce overall lag. Table 6.3.8 gives the average, minimum, and maximum computation time used by the motion controller while under simulation. A single trajectory is tested, with the motion controller called once for each of the 13900 data points. The experiment is run on a standard desktop computer system, with a 3.00 GHz CPU, 1 GB of RAM, and running the FreeBSD operating system. This system is comparable to the computer system that RMCD will be running on within Emulab Mobile.

6.4 Obstacle Avoidance

Obstacle avoidance using the Virtualized Phase Portrait Method is simulated using MATLAB. A two dimensional, holonomic point robot is modeled in a workspace with oriented rectangular obstacle exclusion zones. The phase portrait for a single obstacle with a nearby goal, generated using the original VPPM is given in Figure 6.58.

VPPM is well suited for robot trajectory generation when obstacle placement is sufficiently sparse. The initial position must be outside of all obstacle exclusion zones,

Average Time	0.000075 seconds
Minimum Time	0.000062 seconds
Maximum Time	0.003280 seconds

Table 6.2. Run time data for motion controller implemented in RMCD.

as well as surrounding zones in proximity to obstacles; specifically on the distal side of obstacles. Figure 6.59 illustrates a successful trajectory generated by VPPM. The initial position is just outside an exclusion zone, as is the goal point. There are no nearby obstacles to send the trajectory into neighboring exclusion zones.

Figure 6.60 shows a case where VPPM fails to generate a trajectory which avoids all obstacle exclusion zones. While the trajectory does reach the goal point, it travels through an obstacle exclusion zone. The trajectory is pushed into this obstacle by an adjoining obstacle with a stronger field. The trajectory is first pushed into a concave region, and instead of terminating in a local minima, is forced through the obstacle with a relatively weaker field.

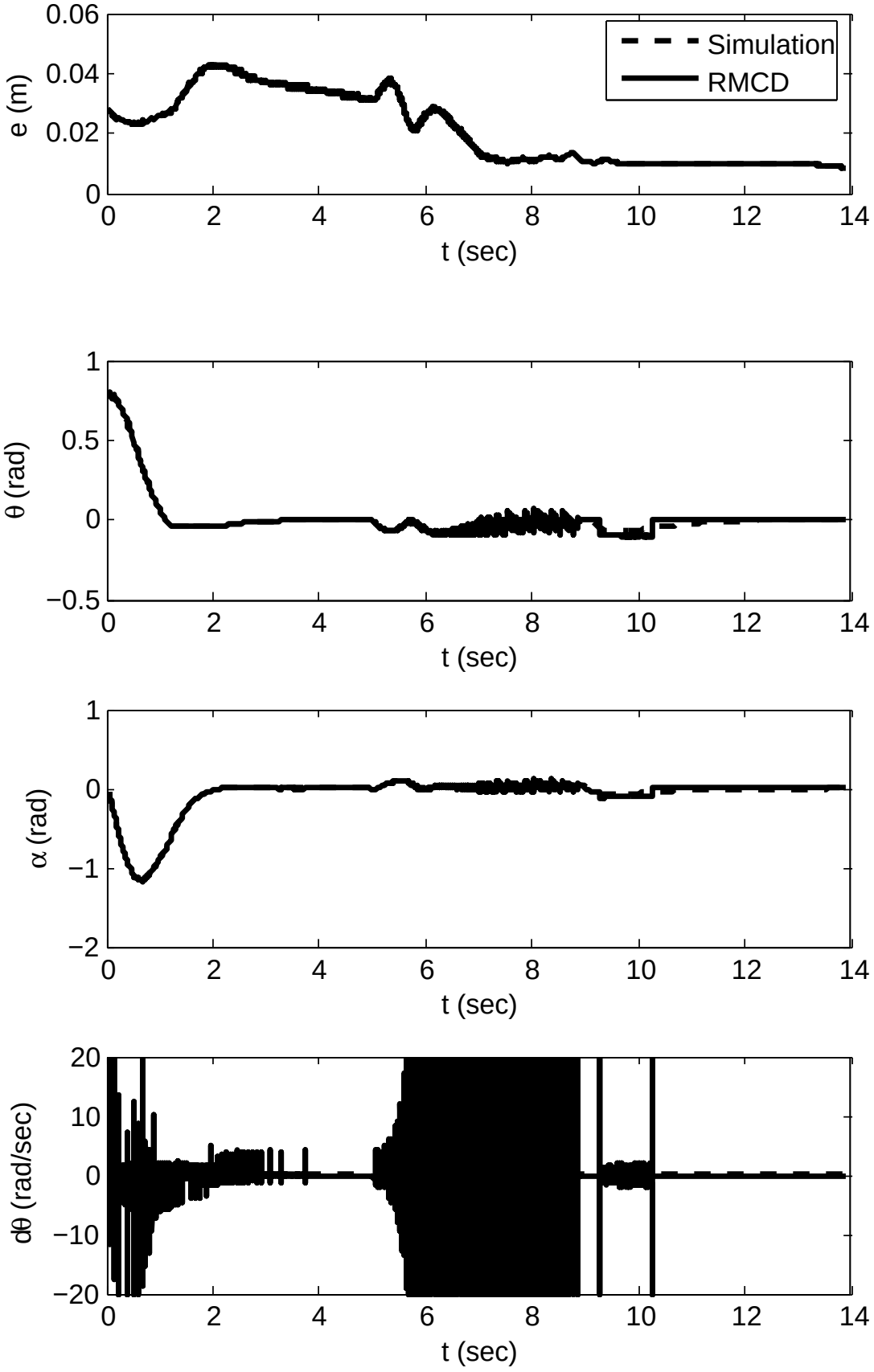

Figure 6.55. RMCD controller implementation, kinematic states.

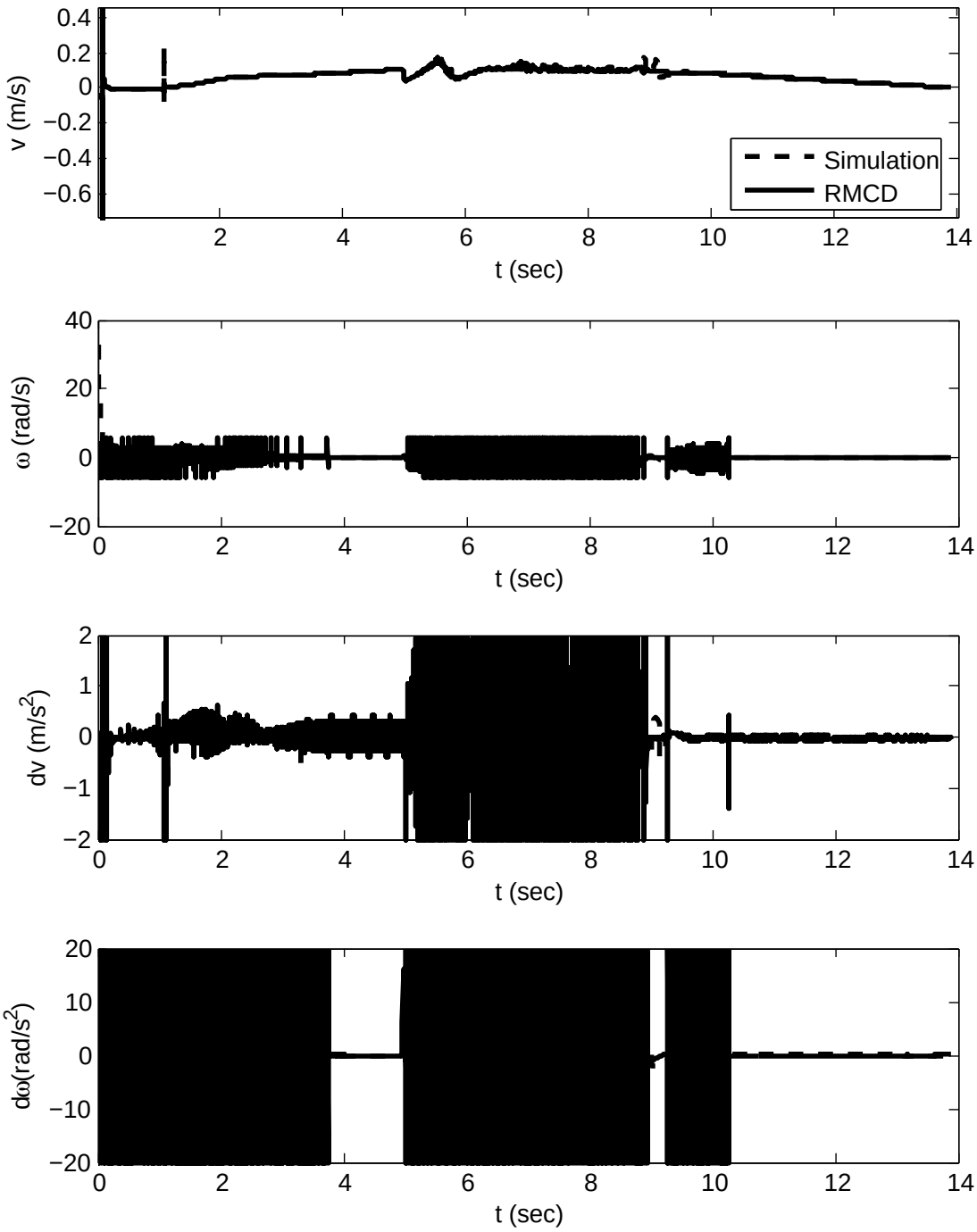

Figure 6.56. RMCD controller implementation, controller output.

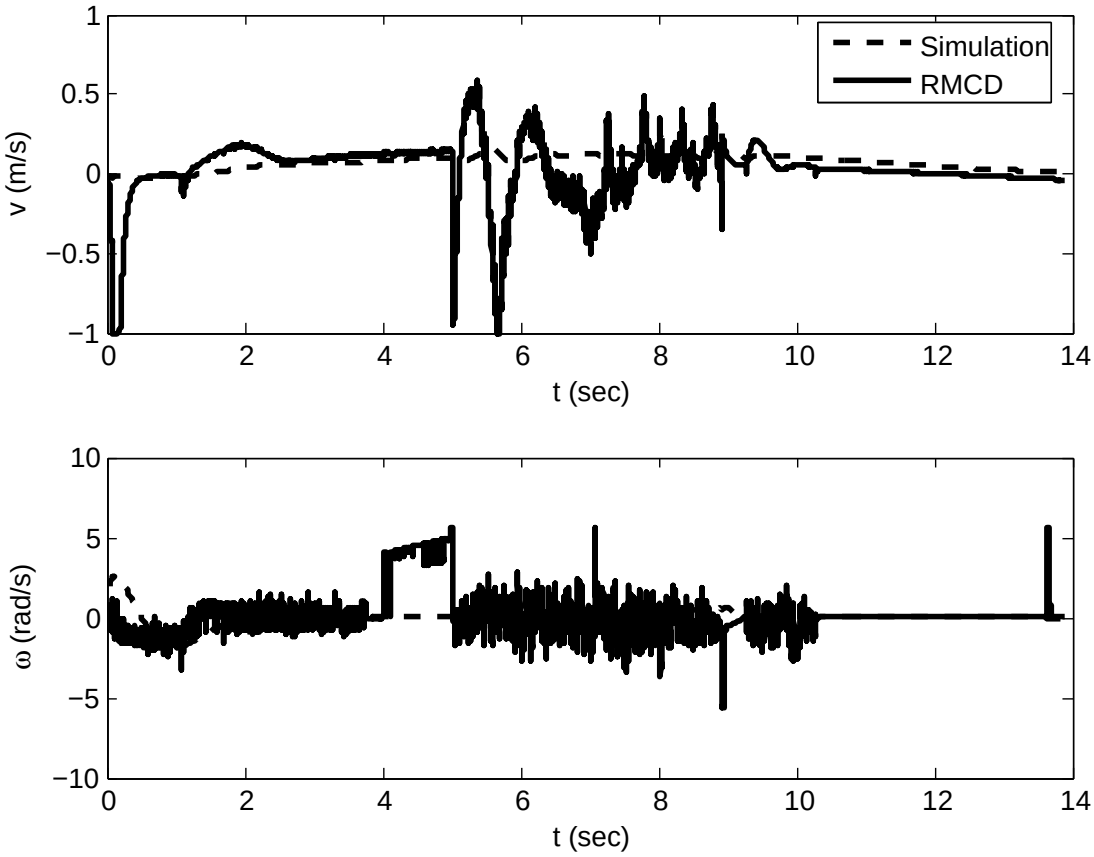

Figure 6.57. RMCD controller implementation, dynamic extension output.

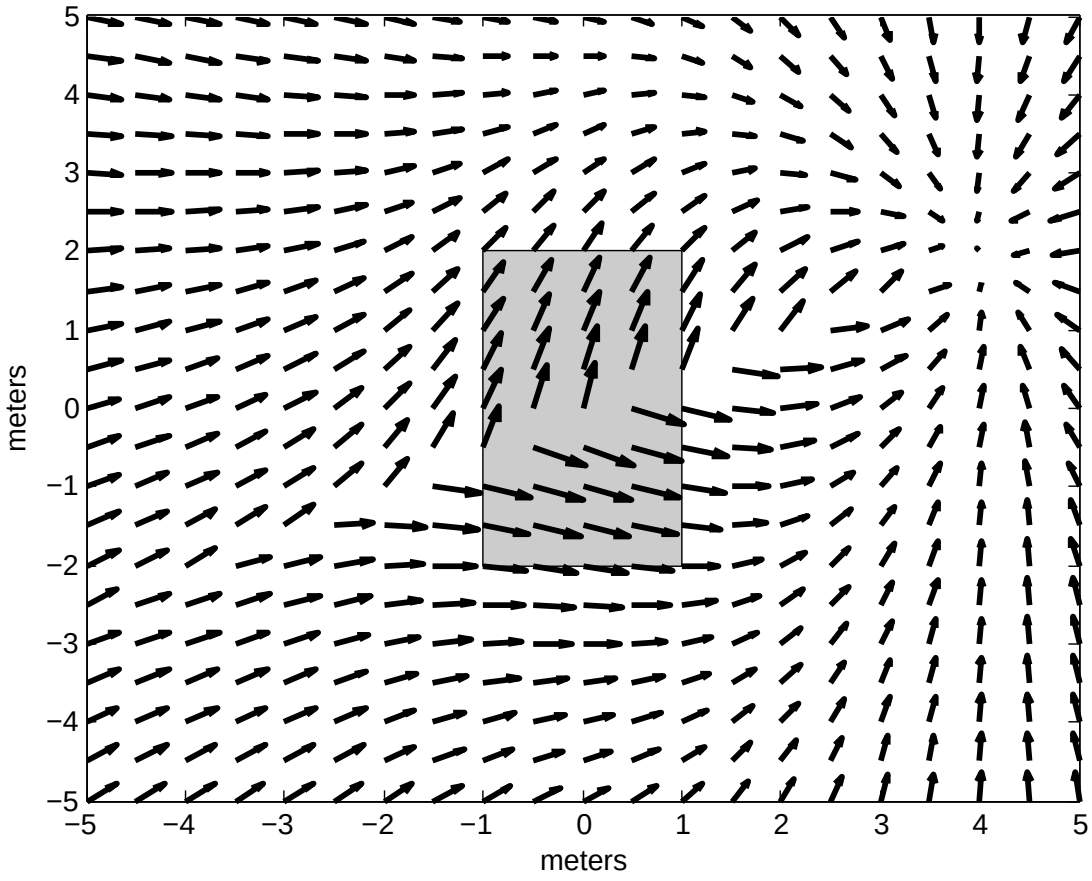

Figure 6.58. VPPM field with a single obstacle.

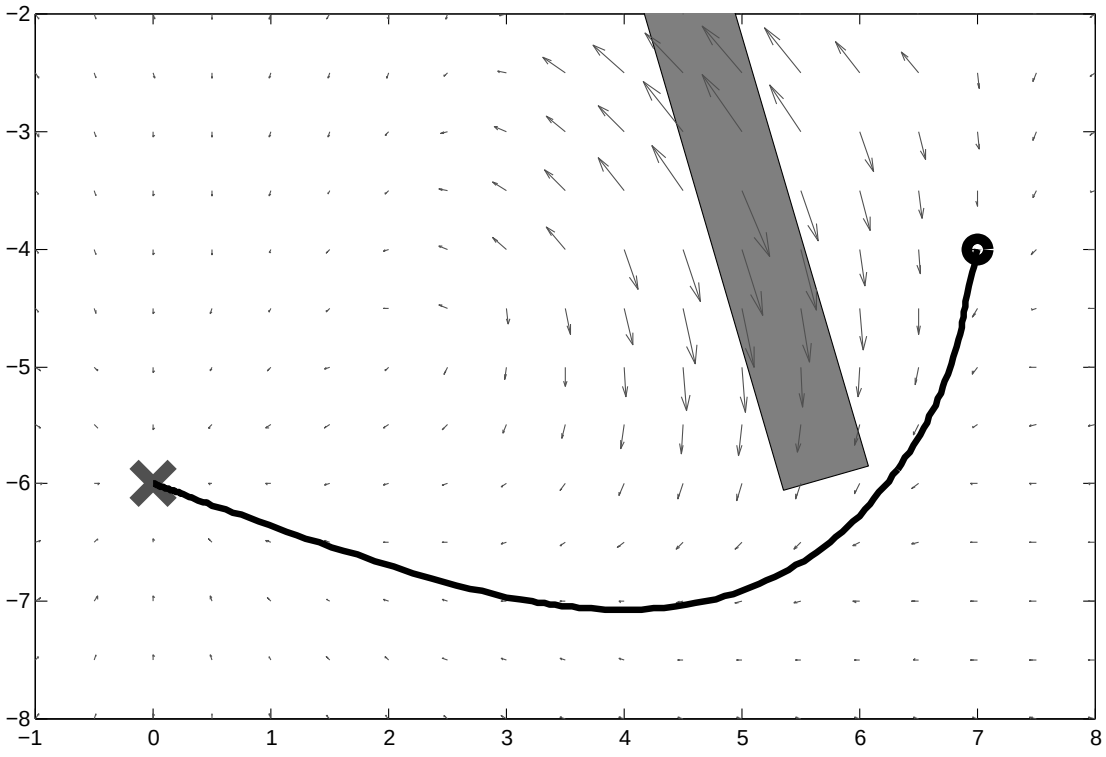

Figure 6.59. VPPM generated trajectory successfully negotiating obstacle filled region.

Similar to the previous failure case, VPPM is shown to fail with closely spaced obstacles with strong fields in Figure 6.61. In this case, both obstacles have the same field strength. A discontinuity is encountered between the two obstacles, once the influence of the larger obstacle drops off sharply. The trajectory enters the exclusion zone of the smaller obstacle as it is forced through a concave region.

The trajectory given in Figure 5.1 in Chapter 5 is a good example of the capabilities of VPPM. The initial condition is chosen to avoid concave regions, and the goal point is located outside of a dense obstacle region. The trajectory is smooth and continuous, and successfully reaches the goal point without colliding with any obstacles.

The velocity profile for this trajectory is presented in Figure 6.62. The desired velocity based on the goal field strength in this example is 0.5 meters per second. Encounters with obstacle fields increases the velocity by roughly 50%, but the velocity remains bounded. In proximity to the goal, the velocity smoothly approaches zero.

The curvature for the trajectory shown in Figure 5.1 is given in Figure 6.63. Curvature in this sense is defined as the inverse of the turning radius. With the exception of jumps

introduced by interference with the rolloff functions of opposing obstacles, the curvature remains smooth and bounded throughout the entire trajectory.

If a trajectory enters the secondary exclusion zone of an obstacle, it is likely to collide with the obstacle. Trajectories can enter these zones by being forced by a strong obstacle in proximity, or by poor initial conditions. The region of the secondary exclusion zone is determined by the size and shape of the obstacle, along with the velocity during approach. Figure 6.64 gives a trajectory with an initial condition outside of a secondary exclusion zone. The trajectory reaches the goal point without colliding with the obstacle thanks to the fact that there was sufficient distance for the repulsive field to deflect the trajectory outside of the secondary exclusion zone during approach to the obstacle. The tangential velocity of this trajectory is shown in Figure 6.65. Curvature is shown in Figure 6.66.

Given an initial condition within an exclusion zone, collision with an obstacle is unavoidable, as shown in Figure 6.67. There is insufficient distance to deflect the trajectory far enough to miss the obstacle boundary. The trajectory proceeds through the obstacle, and then continues on to the goal. The velocity and curvature plots given in Figure 6.68 and Figure 6.69 show that continuity and boundedness is not effected.

The orientation of obstacle field rolloff functions have design tradeoffs. It is important to minimize the impact of the velocity projected on the goal field by obstacle fields. This is to keep velocity bounded, and decrease the probability of local minima being created in proximity to single obstacles. For larger obstacles, field overlays that are perpendicular to the goal field are not parallel to the obstacle boundary. The repulsive field will actually pull a trajectory into an obstacle exclusion zone if the goal is close enough, and the obstacle is large enough. Orienting the field perpendicular to the radial goal line through the center of the obstacle minimizes this effect, but still fails to solve the problem.

In conclusion, the simulation results presented in this chapter establish that the major components of the motion planning and coordination system perform as desired. With system parameters established, and designs verified, these components are implemented into Emulab Mobile. The following chapters discuss the implementation and testing of the motion planning and control systems in Emulab Mobile.

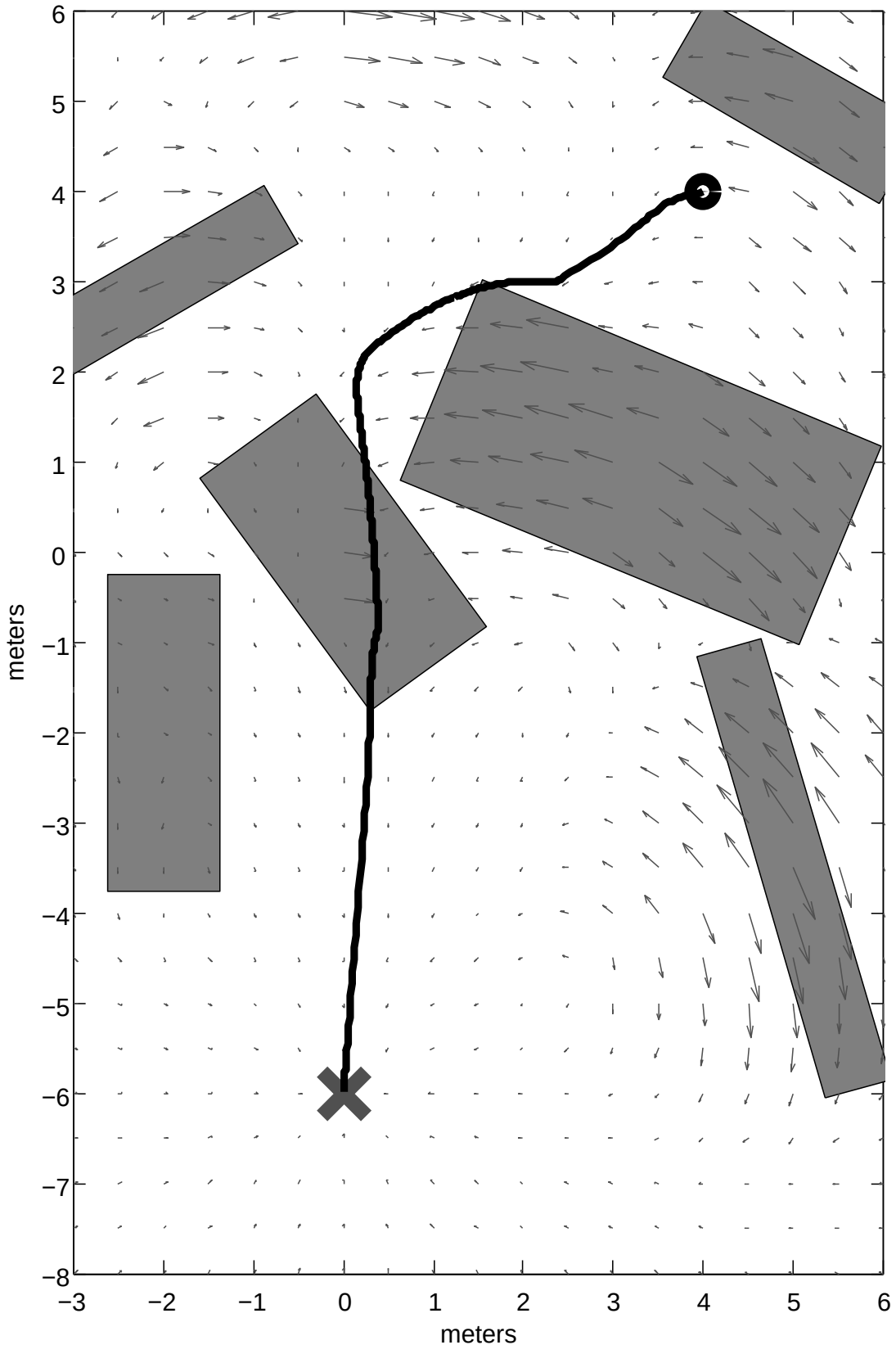

Figure 6.60. VPPM failing to negotiate obstacle filled region.

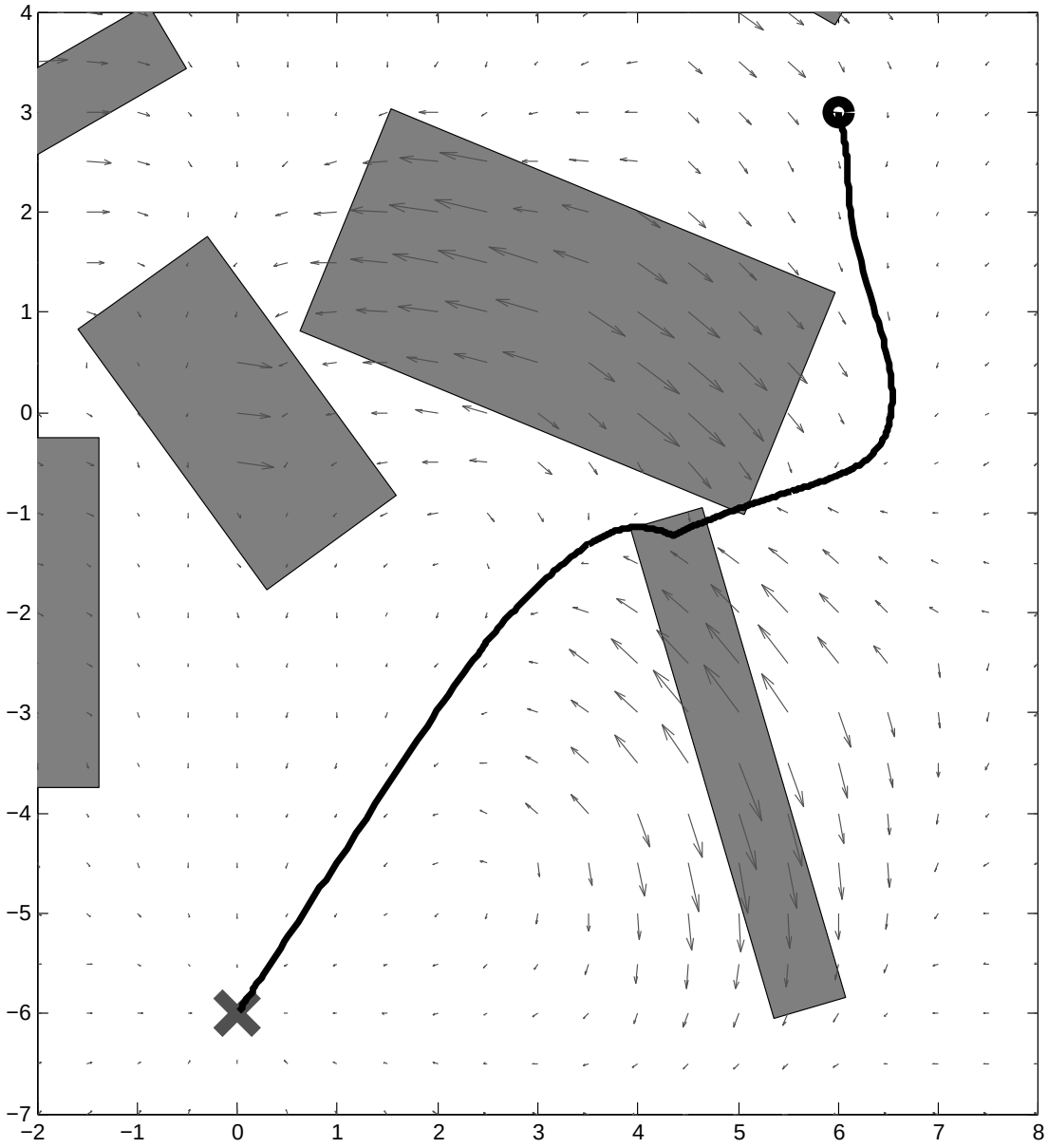

Figure 6.61. VPPM generated trajectory failing in dense obstacle region.

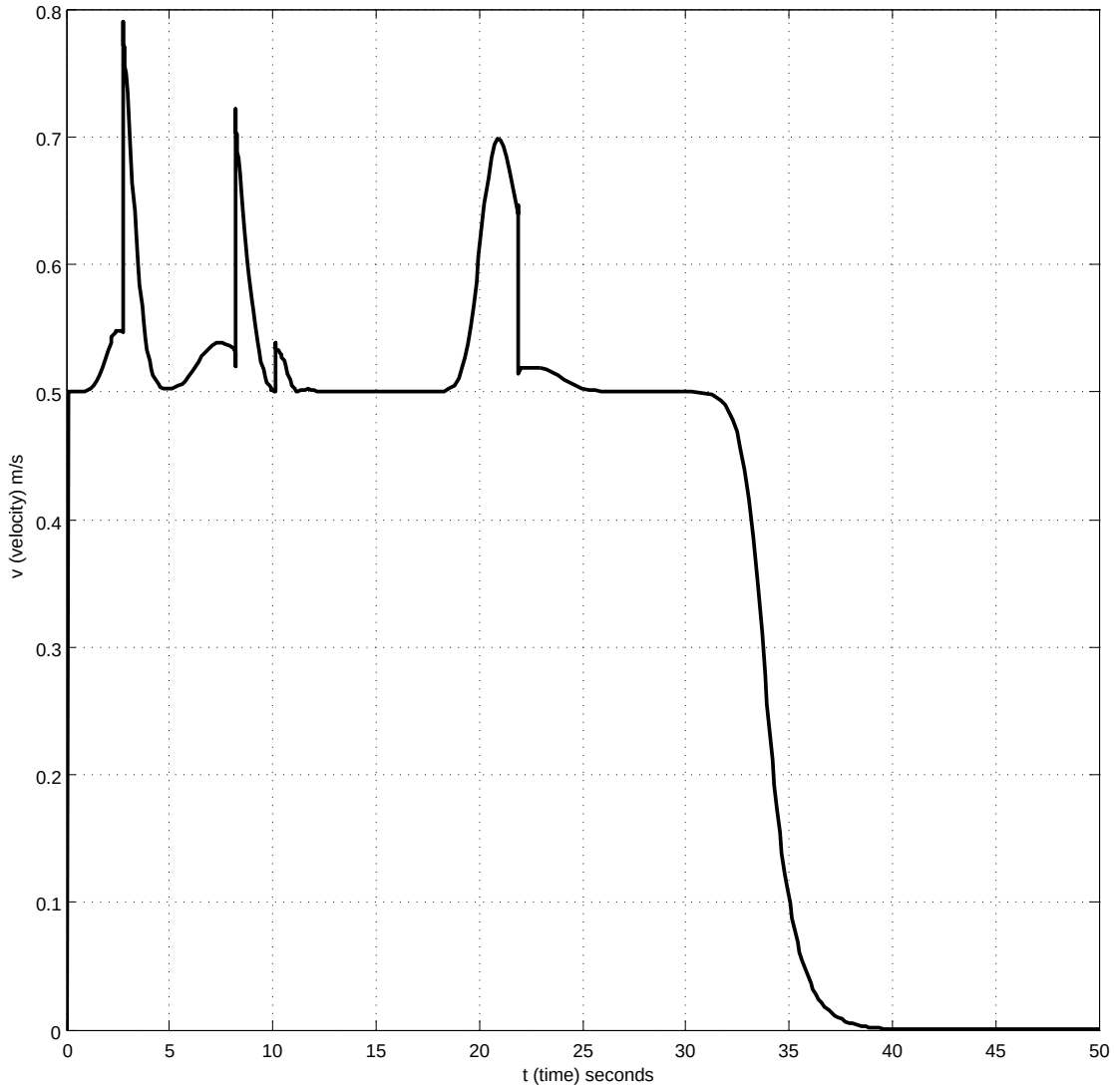

Figure 6.62. Velocity magnitude of trajectory given in Figure 5.1

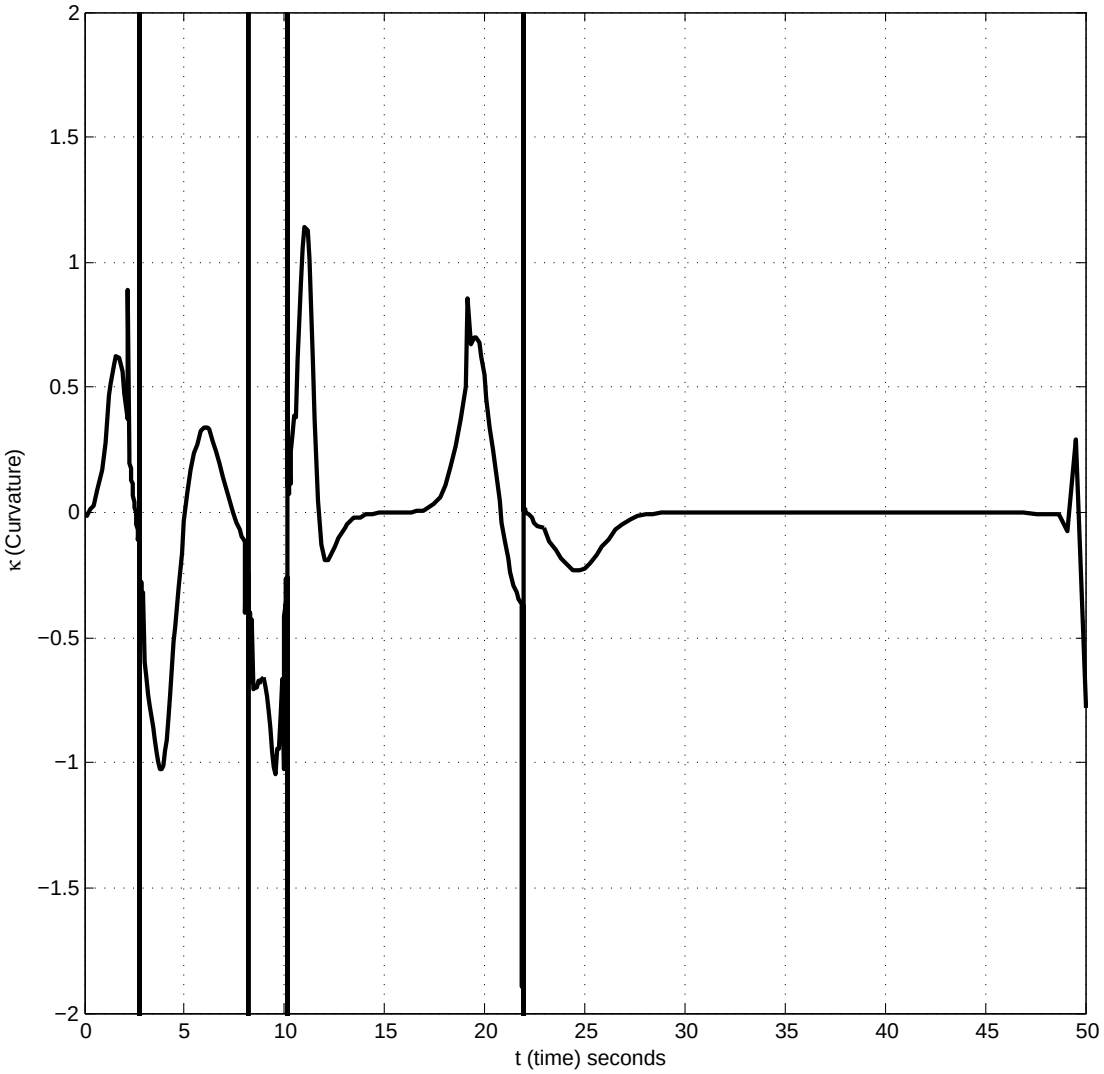

Figure 6.63. Curvature of trajectory given in Figure 5.1

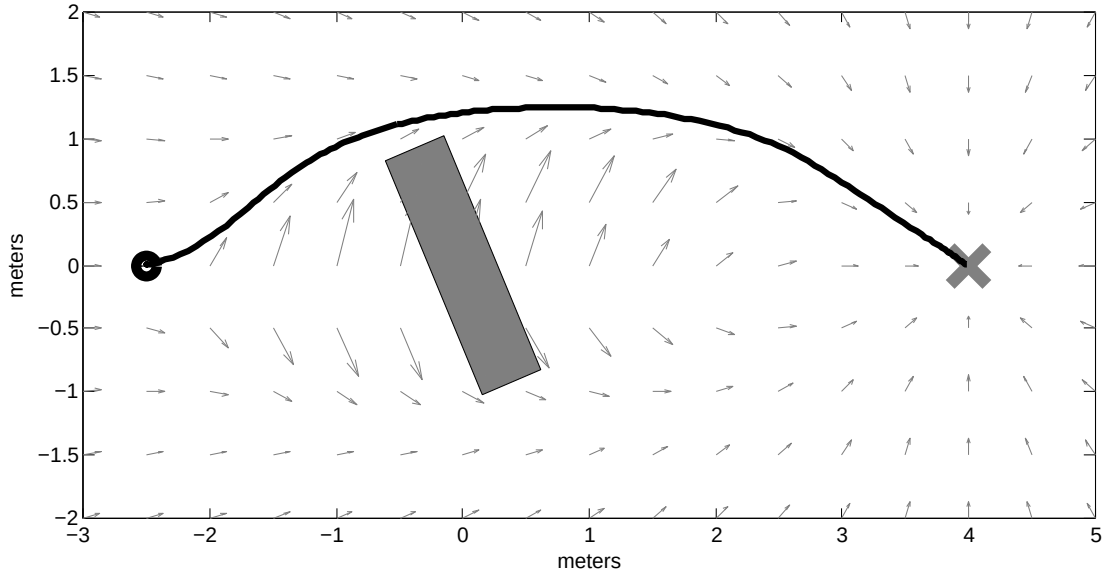

Figure 6.64. Simulated trajectory with an initial position outside of a secondary obstacle exclusion zone.

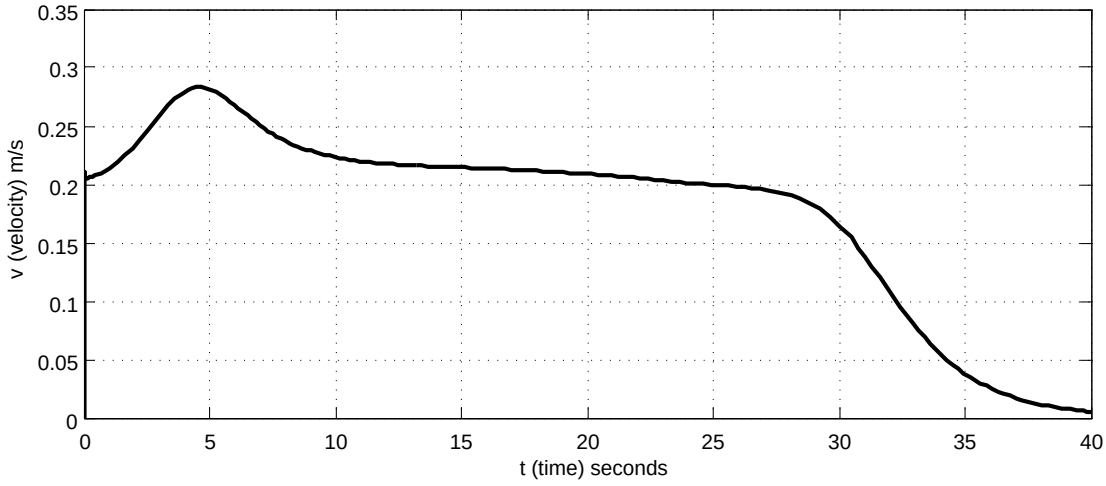

Figure 6.65. Velocity magnitude of trajectory generated in Figure 6.64

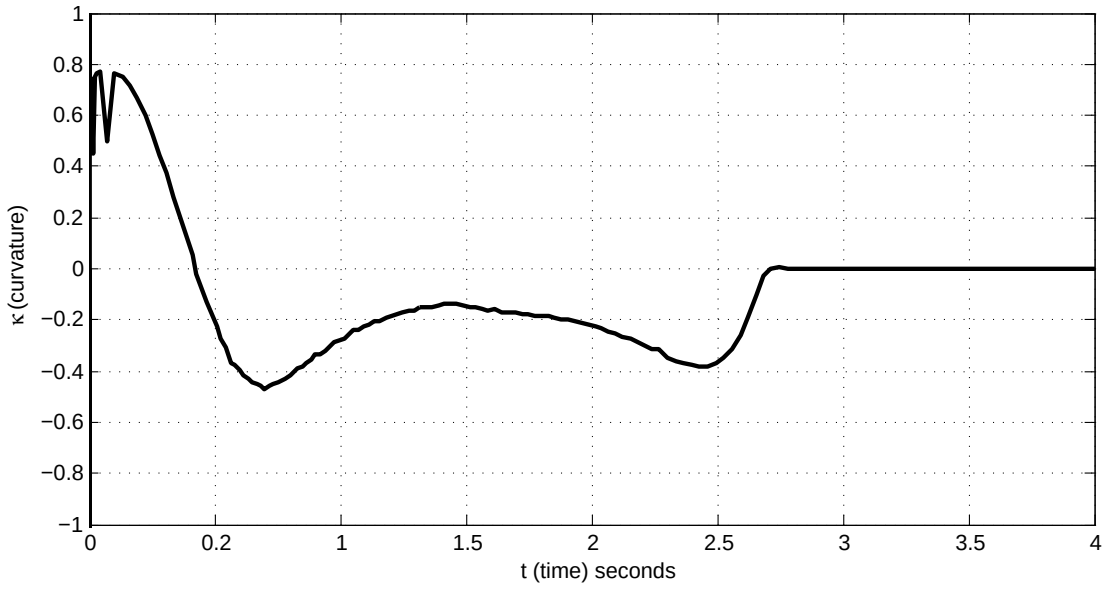

Figure 6.66. Curvature of trajectory generated in Figure 6.64

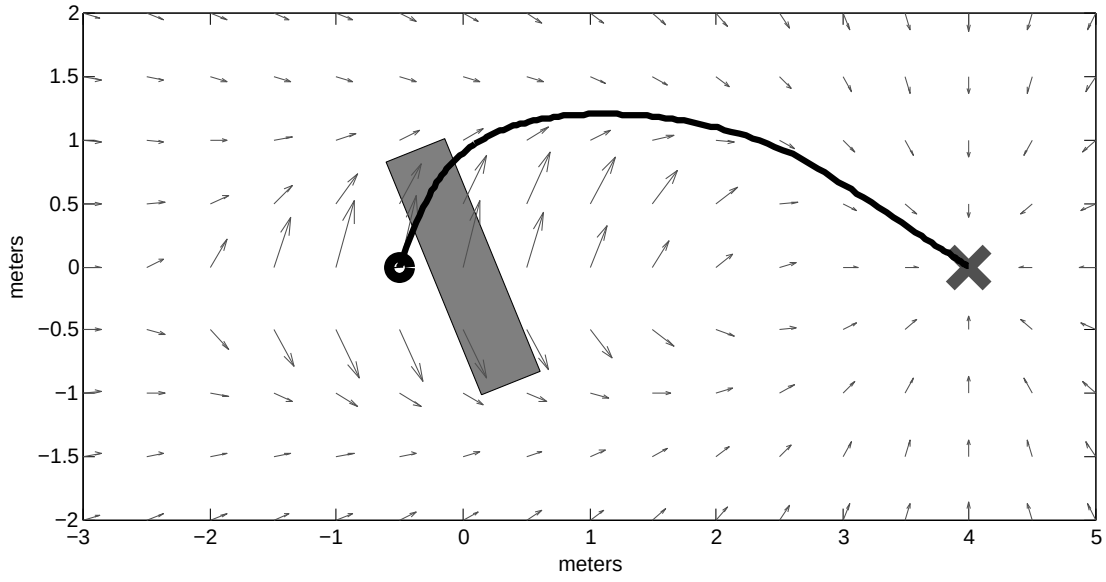

Figure 6.67. Simulated trajectory with initial position inside of a secondary obstacle exclusion zone.

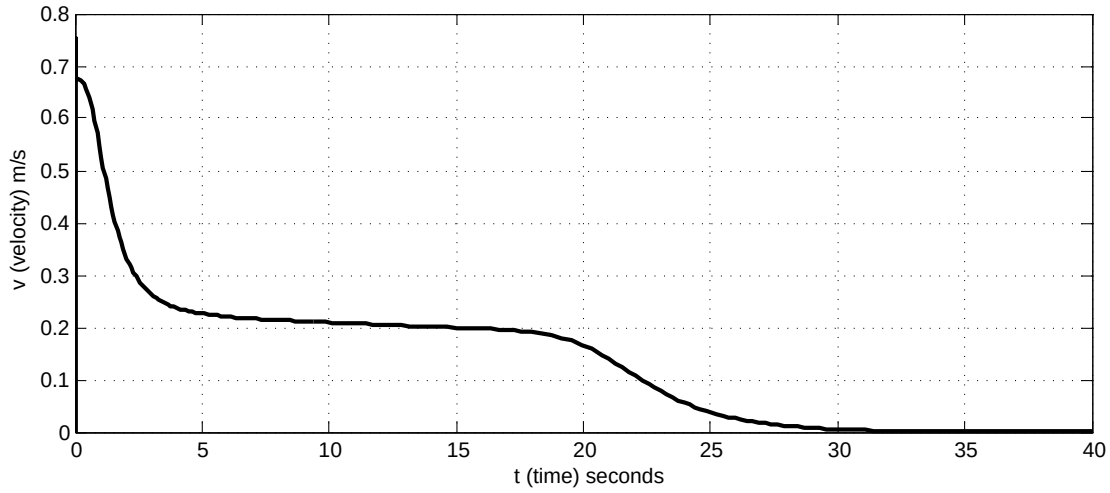

Figure 6.68. Velocity magnitude of trajectory generated in Figure 6.67

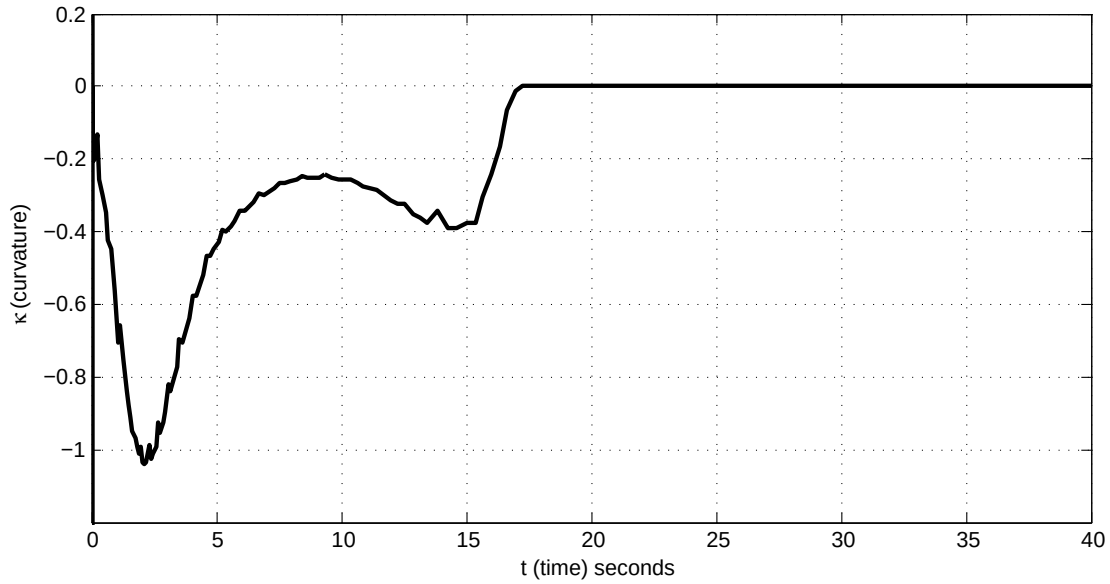

Figure 6.69. Curvature of trajectory generated in Figure 6.67

CHAPTER 7

IMPLEMENTATION

This chapter discusses the implementation and integration into Emulab Mobile the components presented in previous chapters. Primitive motion is the baseline system to be implemented. With point to point motion established on the testbed, state feedback control is then integrated. The posture stabilizing controller is first included, with trajectory generation and the trajectory tracking controller integrated once state feedback control was established as feasible using Emulab Mobile.

To allow for the implementation of state feedback control, system parameters of Emulab Mobile must be verified and measured. The controller implementations must be computationally fast enough to calculate wheel velocity commands before new localization data is received. The sample rate of the state feedback must be quantified, and shown to have a low variance. System identification is performed on a robot to obtain the time constants for the system with a step input.

7.1 Primitive Motion Model Implementation

Point to point motion to a single goal point is accomplished by executing motion primitives in sequence. The primitives *pivot*, *move*, and *pivot* are completed to move a robot to an arbitrary posture in $\Re^2$ Cartesian space. This sequence of primitives is used to construct a meta primitive, referred to as a *goto* command in this research. This command is the core of the iterative goal point progression motion model presented in Section 3.1.

During primitive based motion, a robot self-localizes by using odometry measurements. After each goto command is completed, Emulab Mobile checks positions using the overhead camera localization system. If the position achieved through local navigation using odometry is not coincident with the goal in the global reference frame, subsequent goto commands are issued until the robot is sufficiently close to its intended goal.

Position error becomes significant over long distance moves. As such, goto commands are limited to a displacement of one and a half meters. This allows visual localization to correct robot positions at regular intervals, which minimizes final position error. A velocity value may be configured for each primitive. For *move* commands, velocities as high as 0.8 meters per second have been reliable during testing. Care must be taken not to exceed a stall threshold built into the low level motion controller. The threshold is increased to allow the robots to operate on carpet.

7.2 Posture Stabilizing Controller Implementation

The implementation of the posture stabilizing controller requires significant modifications to Emulab Mobile. The robot control system is extended to support continual state feedback for nonlinear motion control. Wheel stall thresholds still apply as in the primitive motion model, but need to be increased further because of larger accelerations commanded by the state feedback controller.

The posture regulator does not suffer any problems with lower the sampling rates associated with the localization system of Emulab Mobile. This is because of the absence of a moving goal or reference frame.

.

7.3 Trajectory Generator Implementation

The trajectory generators introduced in Chapter 3 are implemented in MATLAB simulations, and integrated into RMCD.

In the parametric line and arc trajectory generator implementation, a full trajectory dataset is calculated upon receipt of a *goto* command. This parametric trajectory data is stored, and later recalled when the motion controller is activated. Waypoint data points are input from a text file configured when starting up RMCD.

Trajectory data parameterized by the trajectory generator is required at nonregular intervals. The motion controller is executed only when localization data is forwarded to RMCD. In order to send an accurate reference trajectory data set, linear interpolation is used.

7.4 Kinematic State Feedback Trajectory Tracking Controller Implementation

The kinematic state feedback trajectory tracking controller presented in Section 4.4 is implemented in the C programming language, and then integrated into RMCD. The controller is implemented as its own suite of functions, allowing it to be removed from RMCD without any difficulty. RMCD must be configured at run time to call the trajectory tracking controller. It defaults to primitive motion when not instructed to use kinematic state feedback control.

7.4.1 Program Structure

The implementation of the trajectory tracking controller is split into several main functions. The entire motion controller is in a single function called by RMCD, named *kc_main*. This function takes as input the current robot position sent by VMCD, the current data point on a parametric reference trajectory, and parameters related to the derivative signals. Wheel velocities are returned as output, which are sent as commands to the robot by RMCD.

The *kc_main* function first executes a Cartesian to Polar state transformation function, *kc_cart2pol*. Controller parameters and gains are then passed to the core controller in *kc_controller*. The controller velocity commands are then sent to the dynamic extension in *kc_dynamic_ext*. The resulting velocities v and ω are then transformed into wheel velocities v_L and v_R, and then sent to the robot.

The control law (4.39) and (4.42) is implemented in the *kc_controller* function. Polar state data, gains, and controller parameters are input. Controller velocity commands v and ω are output.

The dynamic extension, discussed in Section 4.4.2, is included in the function *kc_dynamic_ext*. Helper functions *kc_dynamic_ext_solve* and *kc_dynamic_ext_func* are used to solve the differential equations.

The Cartesian to Polar state transformation is handled by *kc_cart2pol*. Cartesian states are taken as input, and Polar system states are output. The unwrapping of θ and α is done in this function.

7.4.2 System Parameters

The controller parameters given in Table 7.1 must be properly tuned in order to assure stability and achieve the desired performance of the system. This is accomplished

through an understanding of the effects of the various parameters on the system, coupled with rigorous analysis of system stability criteria. The stability analysis of the discretized system is discussed in Section 6.3.4.

There are several design tradeoffs when considering controller parameters and gains. The response of the dynamic extension must be faster than the main controller response, or instability will result. If the time constants of the dynamic extension are too far below the sampling frequency of the motion controller implementation, the response will become unstable.

If the parameter ϵ is decreased, the controller will follow the path manifold more aggressively. An increase of k_1 with cause the error, e to converge faster, and an increase in k_2 causes the controller to steer towards the path manifold more aggressively. Higher values of k_v and k_c increase the response of the dynamic extension, passing through v and ω with less filtering.

Controller output limits for the trajectory tracking controller experimental results presented in Section 8.1 are given in Table 7.2.

7.4.3 Phase Angle Unwrapping

The Cartesian to Polar transformation function uses the *atan2* trigonometric function to calculate the Polar state θ from x, y, x_r, and y_r. This function returns values in the range of $\pm\pi$. Values of θ and α rise or fall through different phases multiple times. A discontinuity results when these values are not unwrapped. For example, at one arbitrary timestep, $\theta_i = \pi - \epsilon$, where ϵ is a small value. At the next timestep, as θ increases, $\theta_{i+1} = -\pi + \zeta$, where ζ is another small value. The value of θ jumps by $2\pi - (\epsilon + \zeta)$. Unwrapping θ results in the jump being the true value of $\epsilon + \zeta$.

Gain	**Description**
r	Path manifold radius
ϵ	Small perturbance to prevent discontinuity
k_1	Controller gain, v
k_2	Controller gain, ω
k_v	Dynamic extension gain, v
k_c	Dynamic extension gain, ω

Table 7.1. Motion controller: system parameters.

7.4.4 Numerical Differentiation

The ω portion of the controller requires the derivative of the Polar state θ, and the dynamic extension requires the derivatives the controller output velocities. As these states can not be measured directly, and there are no observers in the system, the measured states must be differentiated numerically.

Numerical differentiation is accomplished in RMCD by fitting a second-order Lagrange interpolating polynomial,

$$f'(x) = f(x_{i-1})\frac{2x - x_i - x_{i+1}}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})} + \tag{7.1}$$

$$f(x_i)\frac{2x - x_{i-1} - x_{i+1}}{(x_i - x_{i-1})(x_i - x_{i+1})} + \tag{7.2}$$

$$f(x_{i+1})\frac{2x - x_{i-1} - x_i}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)}, \tag{7.3}$$

to the v, ω and θ signals. The first derivative at only the most recent data point is required. As such, $x = x_{i+1}$. This reduces (7.1) to

$$f'(x) = f(x_{i-1})\frac{x_{i+1} - x_i}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})} + \tag{7.4}$$

$$f(x_i)\frac{x_{i+1} - x_{i-1}}{(x_i - x_{i-1})(x_i - x_{i+1})} + \tag{7.5}$$

$$f(x_{i+1})\frac{2x_{i+1} - x_{i-1} - x_i}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)}. \tag{7.6}$$

As is common with numerical differentiation methods, a significant amount of noise is generated in the output. This may cause stability problems with the kinematic controller and its dynamic extension. A low pass filter is needed to smooth the differentiated signals used for successful motion control.

7.4.5 Filtering

To compensate for noise introduced by numerical differentiation of noisy signals, a digital Infinite Impulse Response lowpass filter is designed. The transfer function for the general form of the filter is

$$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}. \tag{7.7}$$

A first order filter of the form,

$$\frac{Y(z)}{X(z)} = \frac{z^{-1}}{1 - z^{-1}}, \tag{7.8}$$

is desired. In the s domain, the transfer function of the filter is given by,

$$F(s) = \frac{2\omega_n \pi}{1 + 2\omega_n \pi}. \tag{7.9}$$

A desired corner frequency, $\omega_n = 10$ Hz is chosen, as it is one third of the frequency of the localization feedback sampling frequency, resulting in,

$$F(s) = \frac{20\pi}{1 + 20\pi}. \tag{7.10}$$

A continuous to discrete transformation is completed, giving a transform in the z domain,

$$F(z) = \frac{0.5115 + 0.5115z^{-1}}{1 + 0.02305z^{-1}}. \tag{7.11}$$

These parameters are used for the IIR filter implemented in RMCD, applied to all numerically differentiated signals.

7.4.6 State Feedback Data Timing

To establish that state feedback data reaches the motion controller, and controller commands reach the robots in sufficient time, experiments are performed to establish the network packet data timings. Localization data is available at 30 frames per second, and the motion controllers implemented in RMCD must be run at as fast a sampling rate as possible. It is also important that there is not too much latency, as lag in the system may cause instability.

Figure 7.1 presents network data packet timing data for a series of experiments where a robot was run under the posture stabilizing controller discussed in Section 4.3. The plot of RMCD packet timings correspond to the sampling frequency of localization data coming from VMCD. The vertical line represents the desired sampling frequency of 30 Hz. Data usually arrives at regular 0.0333 second intervals, the maximum sampling rate constrained by VMCD. Pilot packet timings correspond to the sampling frequency of the nonlinear controller sending wheel speed commands to the Pilot application on the robot. This rate is influenced by the frequency at which the controller in RMCD is called, which in turn is influenced by the sampling frequency of the localization data coming from VMCD. Packet latency for localization data is also given, showing that localization data usually does not lag much more than one third of the sampling frequency.

Measurements of the controller sampling rate during execution of the trajectory tracker are taken. The controller is executed upon receipt of a single packet of data from the localization system. Sampling rate data is collected from two separate experiments. The timestamps of the trajectory data written during controller execution are differenced, creating a list of δt values representing the time elapsed between each iteration of the controller. Figure 7.2 shows a histogram plot of the controller sampling rate for an

experiment run during middle of the day. The controller runs near the specified sampling frequency of 30 Hz, with some variability. A vertical line denotes 33.3 milliseconds in this plot.

The same experiment is run again later in the day. The results of this second experiment are plotted in Figure 7.3. The performance has degraded in this experiment, as shown by the sampling rate for many iterations occuring closer to 40 milliseconds. Many iterations have sampling rates at 70 and 105 milliseconds.

A stability analysis is performed to document the effect of varying sample times on the controller. More details about stability analysis are given in Subsection 6.3.4. The sampling frequency is varied between 0 and 140 milliseconds, which is the range of values seen in Figure 7.2 and Figure 7.2. The maximum magnitude of the Z transform roots is plotted in Figure 7.4. The system is unstable for some values at $T \approx 0.03$ seconds, which is the desired sampling rate. The stability improves at T increases, but goes very unstable for sampling rates $T > 0.128$. For values of $T < 0.128$, stability decreases as T decreases.

7.4.7 System Identification

The Garcia robot is subjected to a step velocity input in order to determine the system time constants needed for discrete stability analysis of the trajectory tracking controller. A step command of $v = 0.5$ meters per second is given for approximately five seconds.

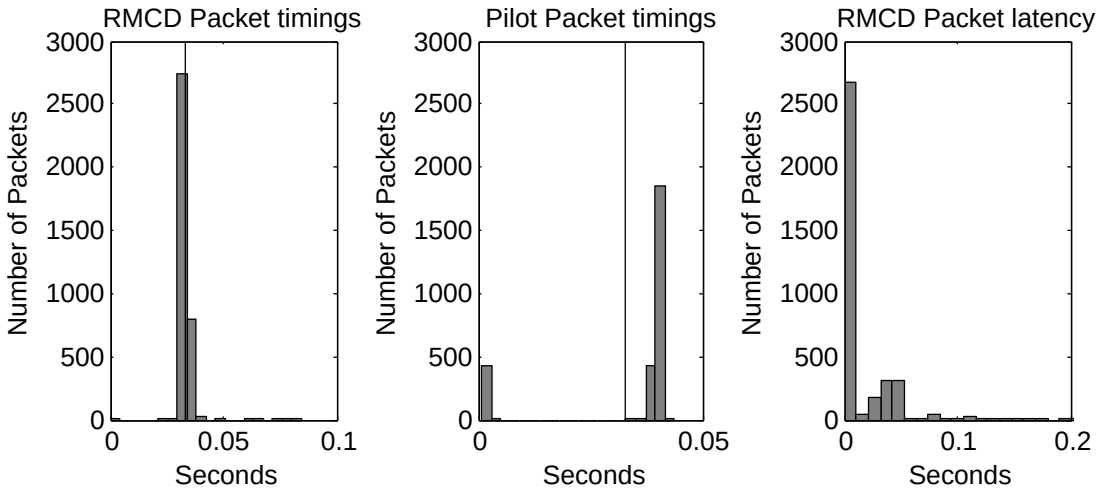

Figure 7.1. Network data packet timings for RMCD and Pilot.

The resulting displacement magnitude, relative to the initial posture of the robot, is given in Figure 7.5. The displacement levels off abruptly due to the robot encountering an obstacle.

The velocity calculated through numerical differentiation of the displacement data is shown in Figure 7.6. A time constant, τ, for the system to respond to a velocity command can be approximated by,

$$\tau \approx t/4, \tag{7.12}$$

where t is the amount of time it takes the system to reach the commanded velocity. From the velocity data presented here, the system reaches 0.5 meters per second in approximately 3 seconds. This results in a time constant $\tau = 0.75$.

The measurements undertaken in this section verify that all system parameters are acceptable for operation of the robots under kinematic state feedback control. The next

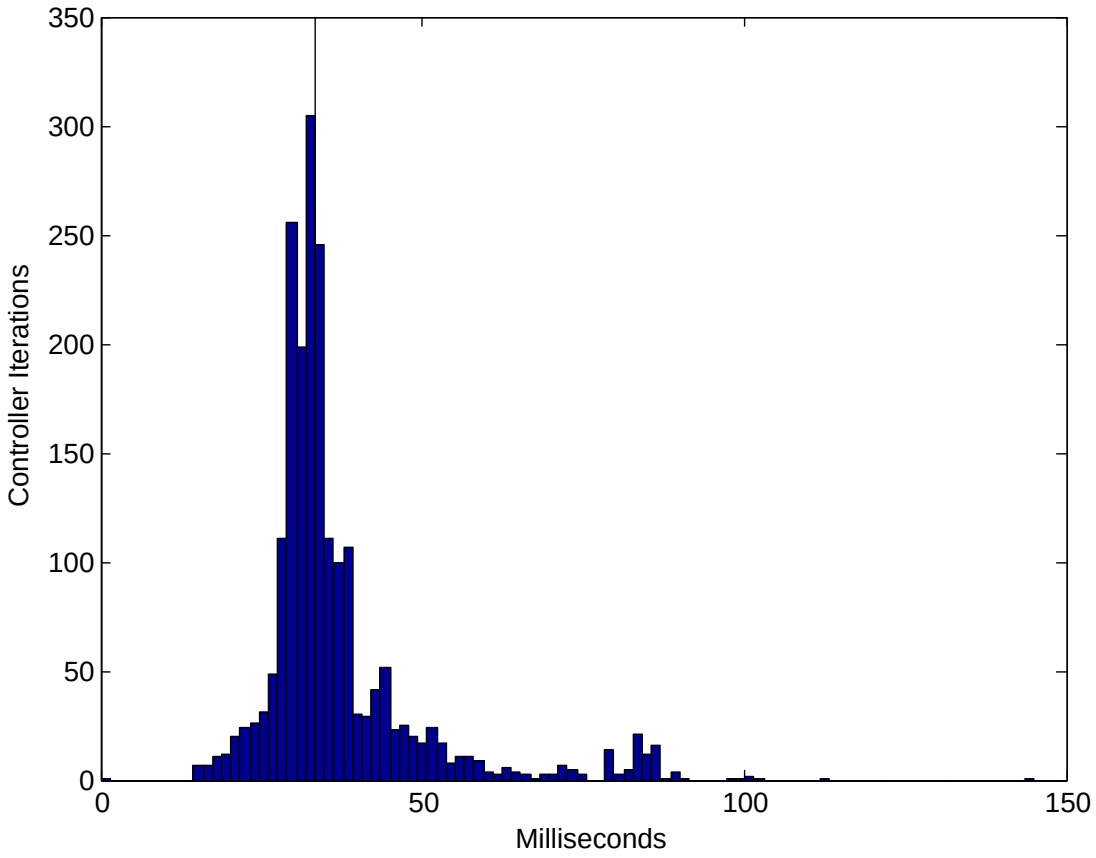

Figure 7.2. Controller sampling rates, experiment 1.

section details the results obtained from operating a robot within the Emulab Mobile workspace.

With the integration of kinematic state feedback control, the performance of Emulab Mobile as a complete system is evaluated. The inclusion of trajectory tracking control allows the robots to execute complex paths, with continual motion.

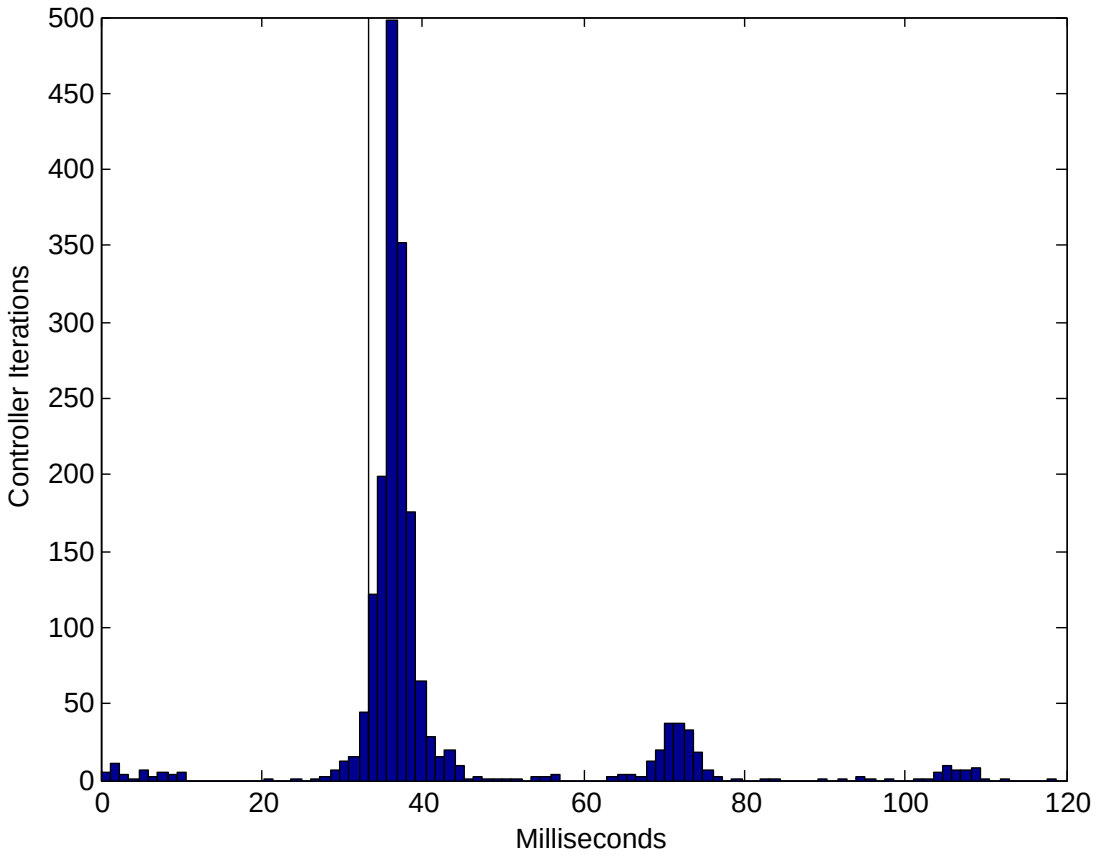

Figure 7.3. Controller sampling rates, experiment 2.

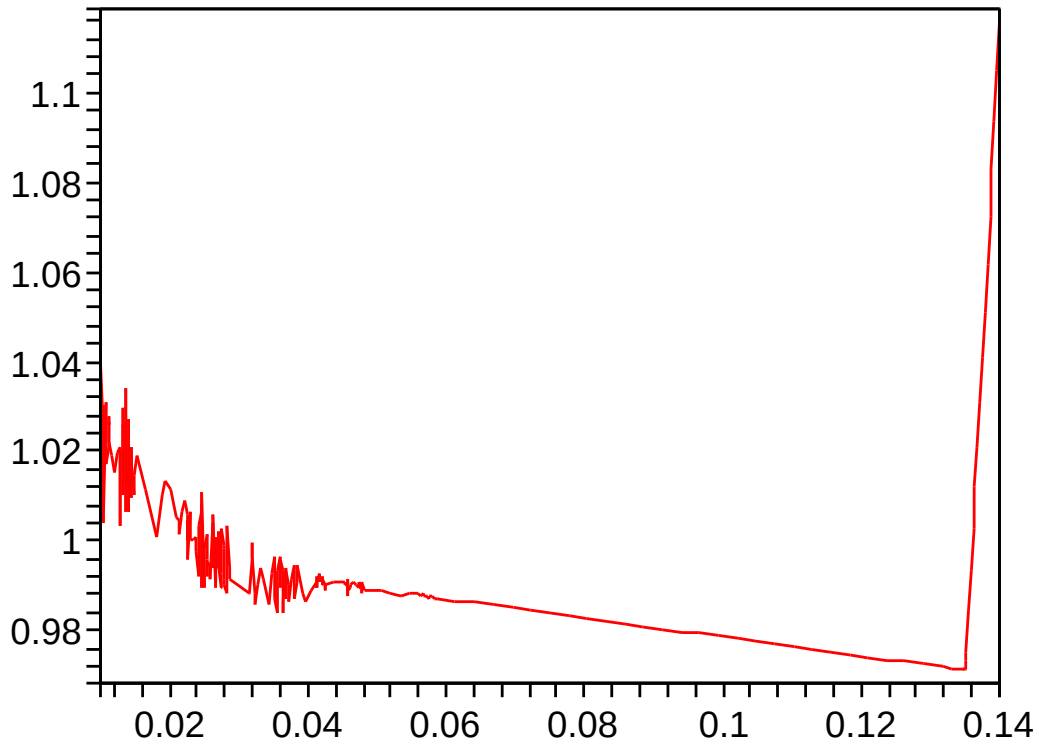

Figure 7.4. Maximum Z transform roots.

Parameter	**Value**
v_{max}	1.0m/s
dv_{max}	2.0m/s^2
ω_{max}	5.6243rad/s
$d\omega_{max}$	20.0rad/s^2

Table 7.2. RMCD controller output limits.

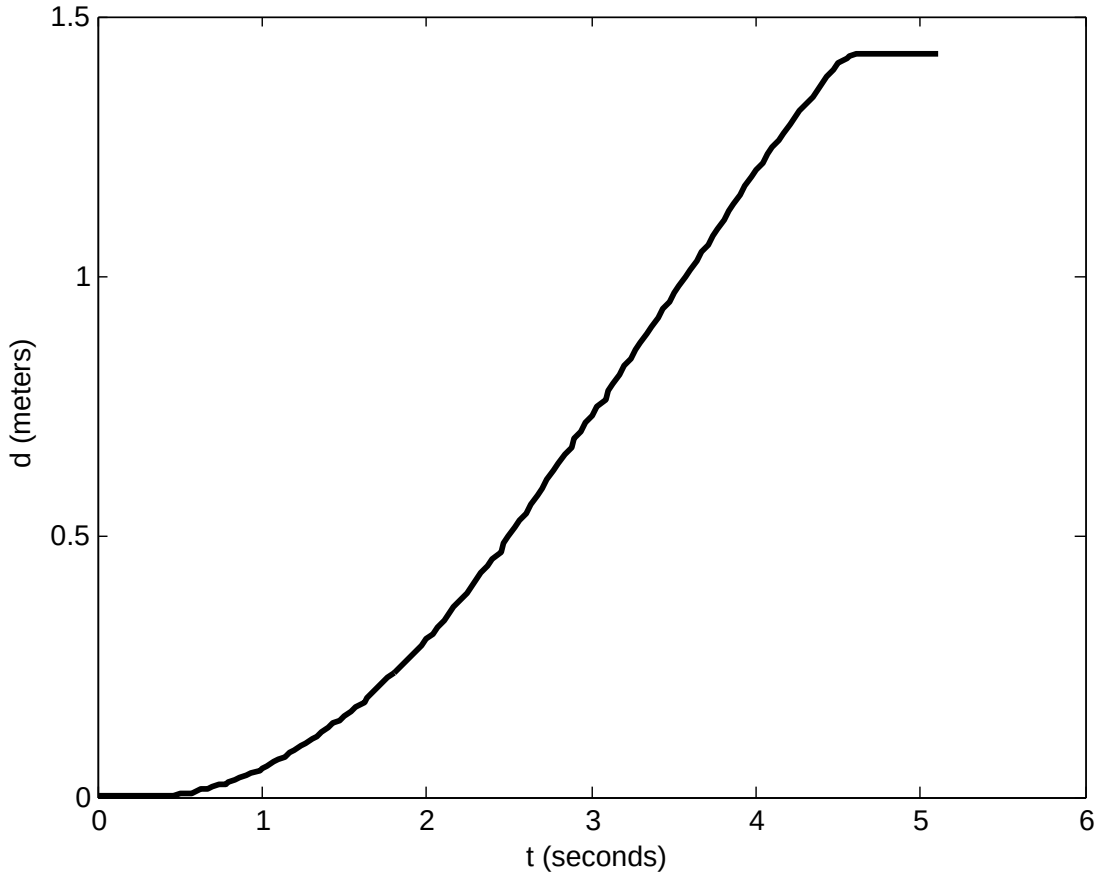

Figure 7.5. Measured displacement magnitude of a step velocity input.

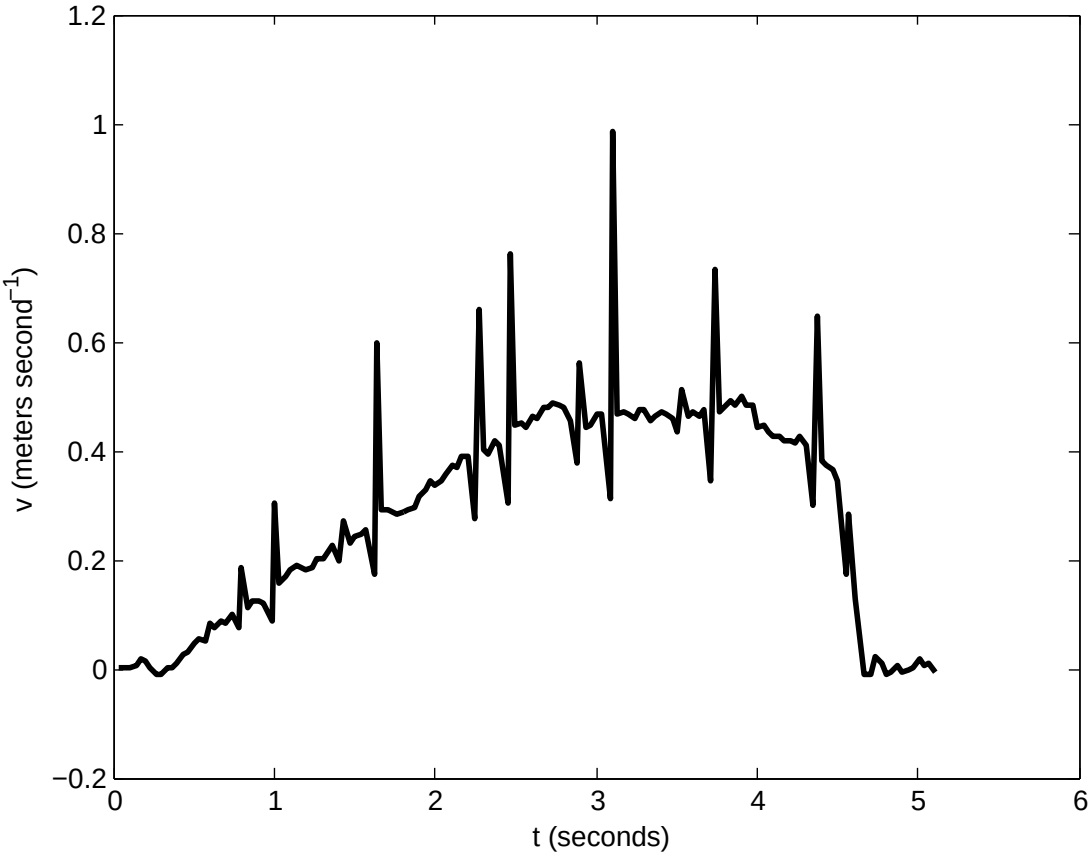

Figure 7.6. Numerically differentiated velocity of a step velocity input.

CHAPTER 8

EXPERIMENTAL RESULTS

The motion planning and control systems discussed in Chapter 3 and Chapter 4 as implemented in RMCD are tested to evaluate their performance and verify their behavior. Experiments are run using Emulab Mobile, with robots operating in the same workspace used by normal users. RMCD is instrumented to log parametric data on robot posture, reference posture, controller velocity commands, and state data. Section 8.1 presents experimental results of the trajectory tracking controller.

8.1 Kinematic State Feedback Trajectory Tracking Controller

A series of experiments are run to evaluate the stability and performance of the kinematic state feedback trajectory tracking controller. Waypoint paths are designed to have a number of curved segments, with a suitable path length to test for accumulating errors. The implementation in RMCD is rigorously tested under real world conditions in this section.

The experiments start out with short line segments without curves, then progress to more advanced trajectories with low reference velocities. After controller parameter tuning for stability and performance criteria, complex paths with higher reference velocities are tested.

All results come from data logged from Emulab Mobile, and indicate data obtained from robots operating in their normal work environment. Trajectories are generated using the system outlined in Chapter 3. The curve type for all experiments is circular arc, as discussed in Section 3.3.

The initial results of the controller obtained shortly after implementation are presented in Subsection 8.1.1. The reference path in this instance is a two colinear line segments, with no curve.

Parameter	Value
R	0.02
ϵ	0.001
k_v	3.0
k_c	0.5

Table 8.1. Controller parameters used for preliminary trajectory tracking experiment.

8.1.1 Initial Trajectory Tracking Experiment: Straight Line Path

A preliminary test run of the trajectory tracking controller in RMCD is performed before parameter tuning. The resulting trajectory of this test is given in Figure 8.1. A straight line reference trajectory is chosen, one meter long, and with a maximum velocity of 0.1 meters per second. The parameters chosen for this experiment are outlined in Table 8.1. The controller eventually fails to track the reference trajectory, terminating the experiment early.

The polar system states are shown in Figure 8.2. The state e in this experiment is larger than the simulated value, as the robot has trouble tracking the reference trajectory. The state $d\theta$ has a large amount of noise compared to simulation.

Figure 8.3 shows the outputs of the main controller, and their derivatives. The experimental results for the output v are similar to the simulation results.

The dynamic extension outputs are given in Figure 8.4.

8.1.2 Straight Line Path with Dynamic Extension Disabled

The measured trajectory of this experiment is plotted in Figure 8.5. The reference trajectory is followed with minimal offset error, though there is overshoot present at the

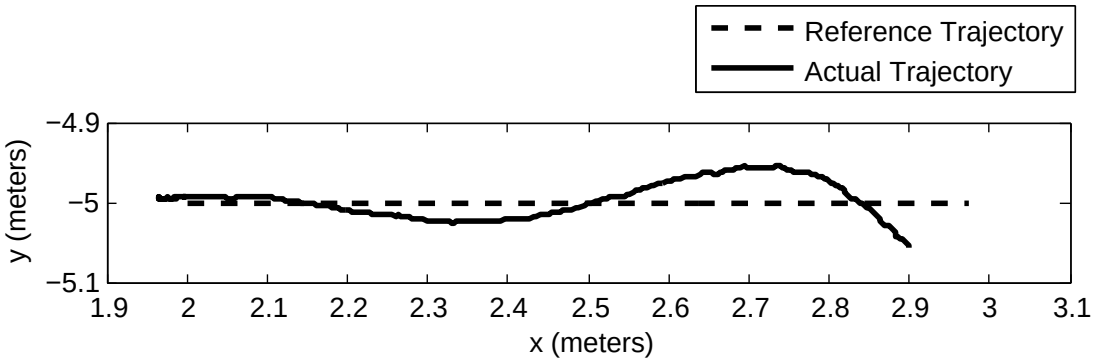

Figure 8.1. Kinematic state feedback controller, resulting trajectory.

r	0.05 meters
ϵ	0.03 meters
k_1	0.3
k_2	0.3

Table 8.2. Controller parameters for straight line and curved path experiments.

end of the run. The reference path starts at $[2.0 - 5.0]$ and ends at $[3.0 - 5.0]$. The controller parameters used for this experiment are in Table 8.2.

The system states are shown in Figure 8.6. The position error is higher than in other runs because of the suppression of the dynamic extension. Error remains below 5 centimeters, which is acceptable.

Figure 8.7 is a plot of the controller commands, which are translated into wheel speeds and then sent directly to the robot. There is noise present in the ω signal, which corresponds to rotational velocity. This is a result of noise in the differentiation of Polar state θ.

8.1.3 Three Segment Path with a Single Curve

Another test run of the trajectory tracking controller is completed, this time added an additional line segment, along with a curve. The curve radius is constant at 0.5 meters, with the initial reference point at $[2.0 - 5.0]$. The same controller parameters are used as in the experiment discussed in Subsection 8.1.2. These parameters are given in Table 8.2. Figure 8.8 illustrates the resultant trajectory, with the reference and actual robot postures represented by the wheels and axles. The dynamic extension is enabled in this run.

The start of the run is illustrated in Figure 8.9. The robot initially converges on the reference trajectory, but lags behind by approximately 5 centimeters. The initial point of the robot is at $[1.99 - 4.99]$, which is an acceptable initial condition for the controller.

The Polar states for this experiment are plotted in Figure 8.10. The derivative of θ, $d\theta$ is noisy near the end of the run, as the states θ and α rapidly increase.

Figure 8.11 contains the velocity commands and their derivatives, as calculated by the controller. The ω signal is noisy, influenced by the noise present in $d\theta$.

The output from the dynamic extension is given in Figure 8.12. The noise in both v and ω is attenuated. The dynamic extension acts as a low pass filter in this case, significantly improving the characteristics of the controller command signals.

Parameter	Value
R	0.2 meters
ϵ	0.03 meters
k_1	0.3
k_2	0.3
k_v	3.0
k_c	3.0

Table 8.3. Controller parameters used for low speed trajectory experiment.

Left and right wheel velocities are presented in Figure 8.13. The data presented here represent the velocity commands calculated by (4.10), not the measured wheel velocities on the actual robot. The noise present in the signals is a result of the noise in the dynamic extension output. As expected when executing a turn in the positive (right hand) direction, the right wheel velocity is greater than the left.

8.1.4 Figure Eight Path: Low Velocity

A more complex reference trajectory is devised for this experiment. The path is in the form of a figure eight, with turn radii of 0.5 meters.

A maximum reference velocity of 0.05 meters per second, with a curve radius of 0.5 meters is used for this experiment. All curves in the reference trajectory are constant radius circular arcs. The path waypoints are $[2.0 - 5.0]$, $[3.0 - 5.0]$, $[3.0 - 4.0]$, $[2.0 - 4.0]$, $[2.0 - 6.0]$, $[1.0 - 6.0]$, $[1.0 - 5.0]$, $[2.0 - 5.0]$, in meters. The dynamic extension is enabled, as is the IIR filter. The configured robot acceleration upon execution of the *null* primitive is 0.6 meters per second squared. The controller parameters are given in Table 8.3. The reference trajectory starts and ends at $[2.0 - 5.0]$, as shown in the above waypoint data.

Figure 8.14 is a plot of the actual robot trajectory, compared to the reference trajectory. The robot tracks the trajectory through all the curves with minimal error. To illustrate the tracking error along the path, a plot of the robot compared to the reference robot at a single instant in time is given in Figure 8.15.

The Polar states for this experiment are plotted in Figure 8.16. There are several discontinuous jumps in θ and α, and e is approximately 4 centimeters through the entire run.

Figure 8.17 shows the corresponding dynamic extension velocity commands for this experiment. The signal v is smooth, with little noise, and rapidly converges to the configured maximum reference velocity of 0.05 meters per second. Noise is present in

the ω, which is a result of the derivative noise from other components of the motion controller.

This experiment illustrates that the trajectory tracking controller is capable of tracking a complex reference trajectory at a lower reference velocity. Position error is still higher than desired, but the controller is stable. Operating the robot at lower velocity is a detriment to performance, and the signal to noise ratio lowers as the velocity lowers.

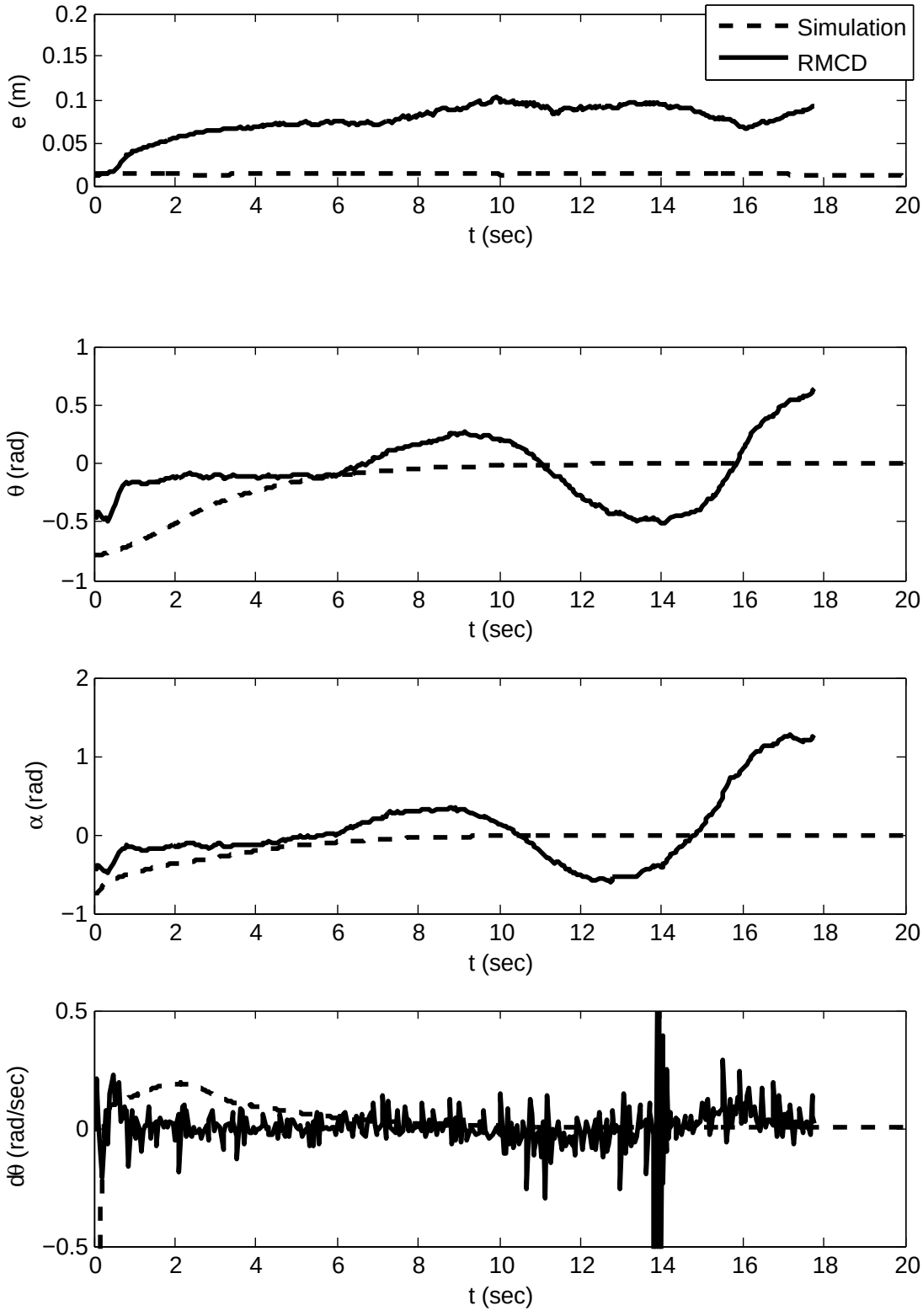

Figure 8.2. Kinematic state feedback controller, polar states corresponding to Figure 8.1.

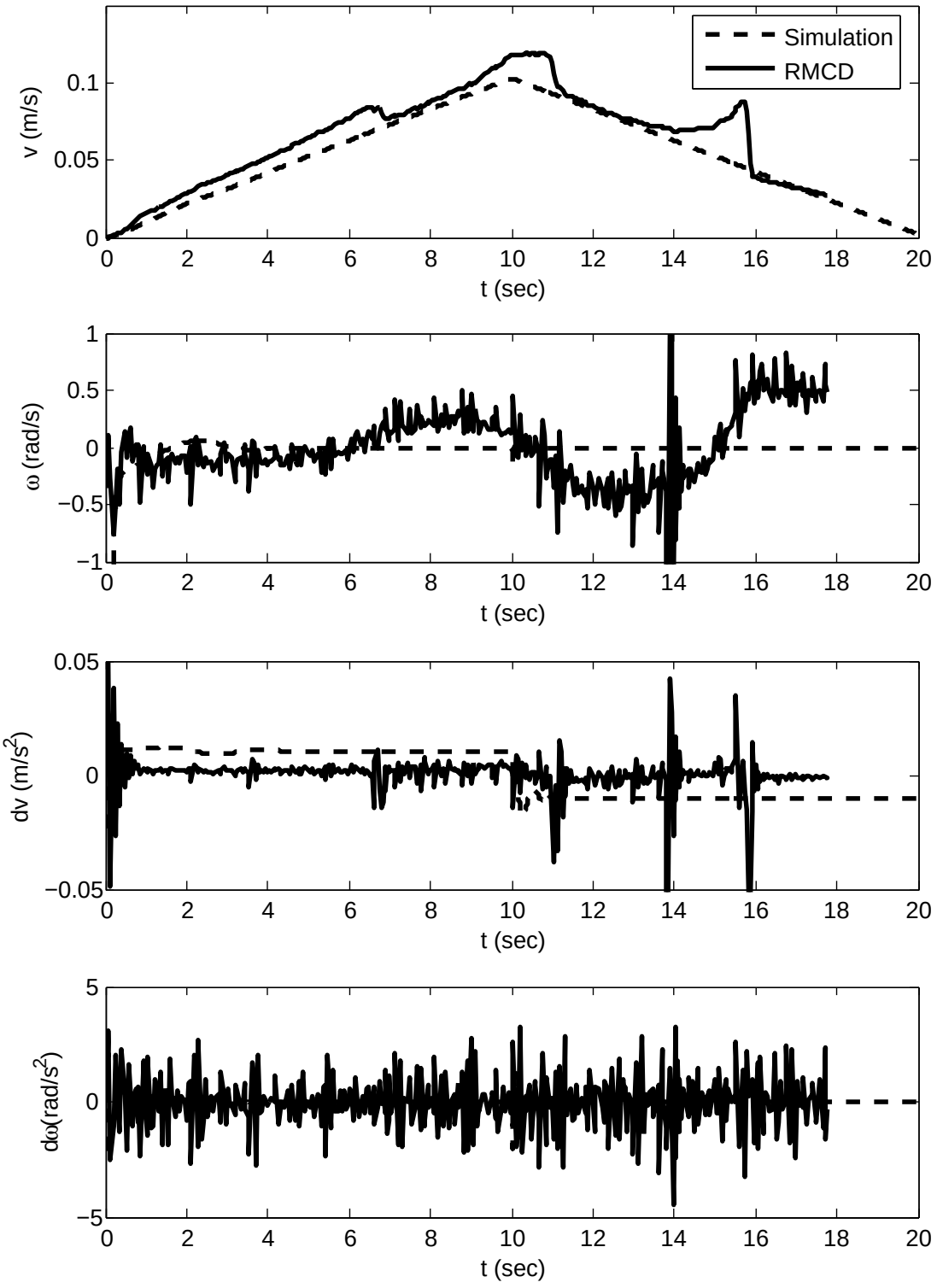

Figure 8.3. Kinematic state feedback controller, controller output corresponding to Figure 8.1.

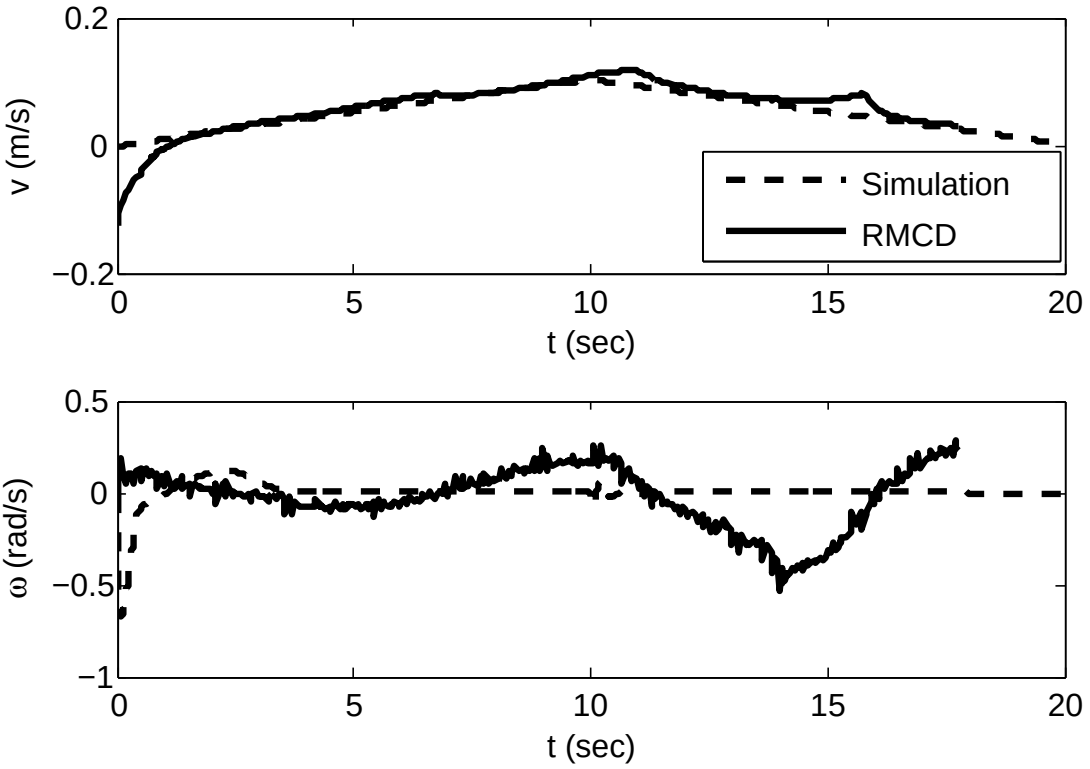

Figure 8.4. Kinematic state feedback controller, motion controller output corresponding to Figure 8.1.

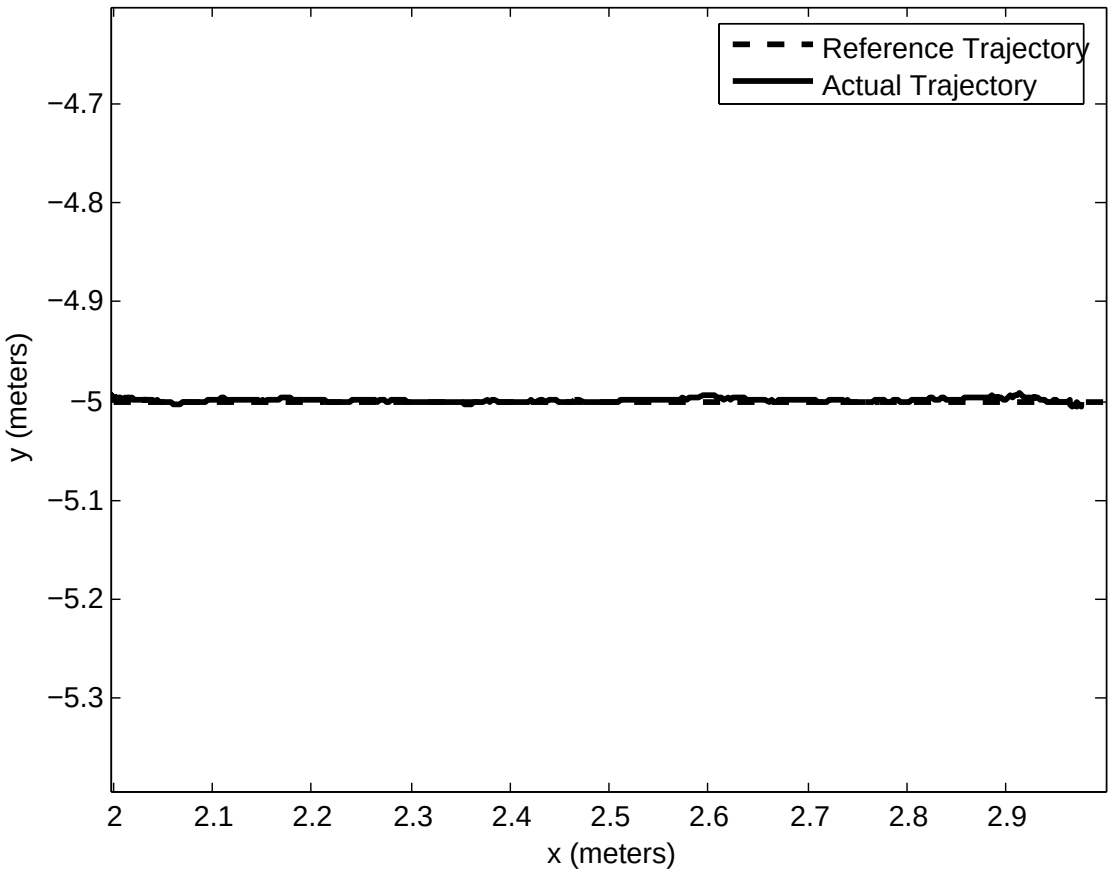

Figure 8.5. Straight line trajectory tracked with dynamic extension disabled.

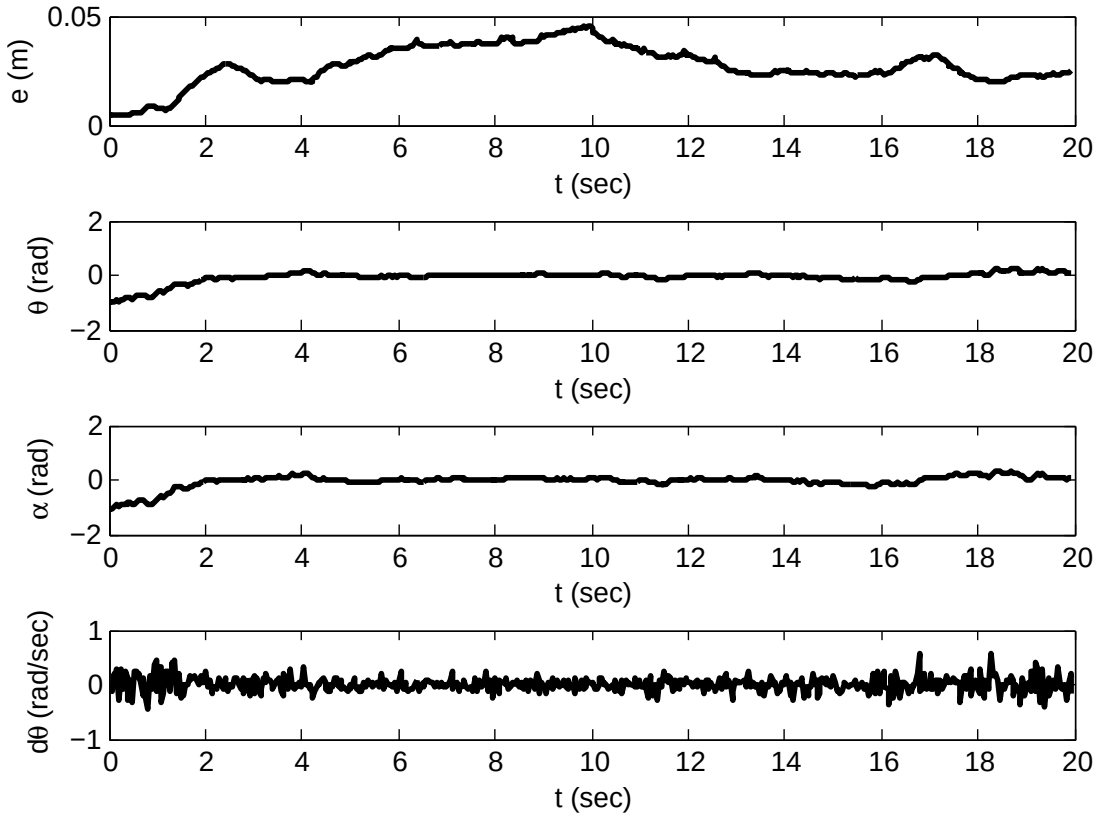

Figure 8.6. Straight line trajectory tracked with dynamic extension disabled, Polar states.

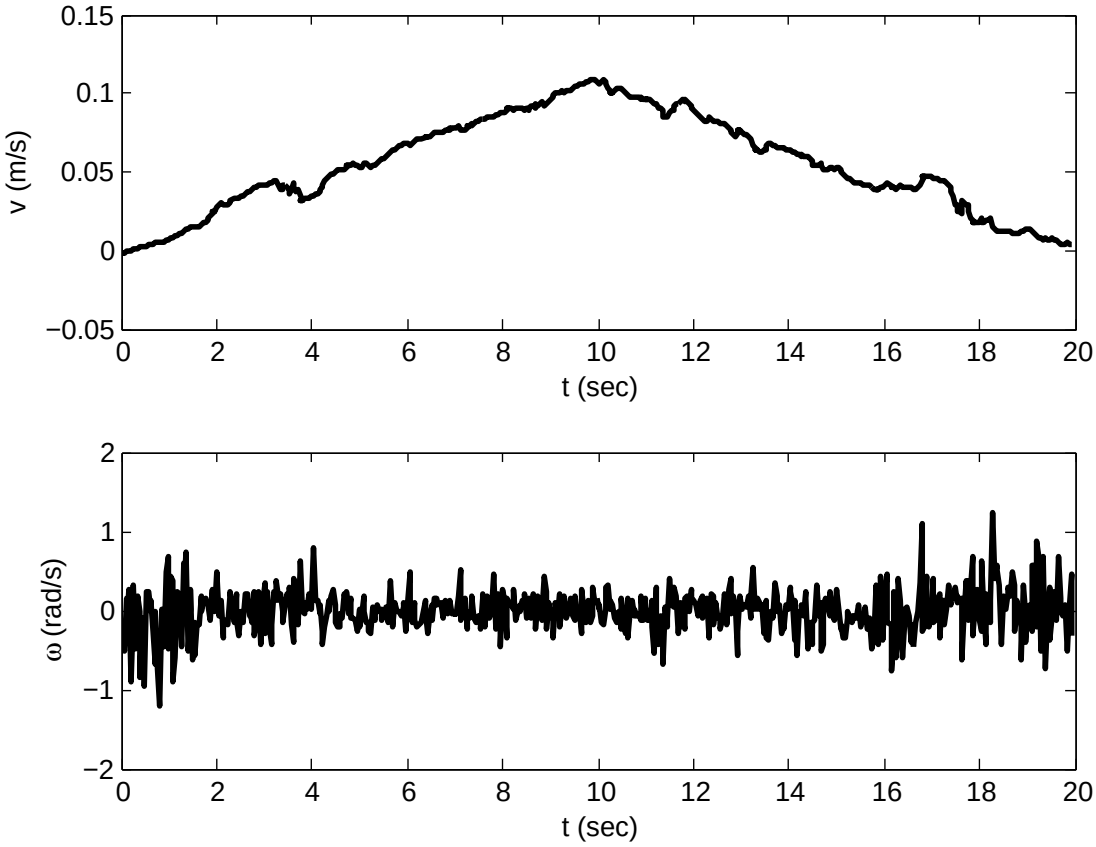

Figure 8.7. Straight line trajectory tracked with dynamic extension disabled, controller velocity commands.

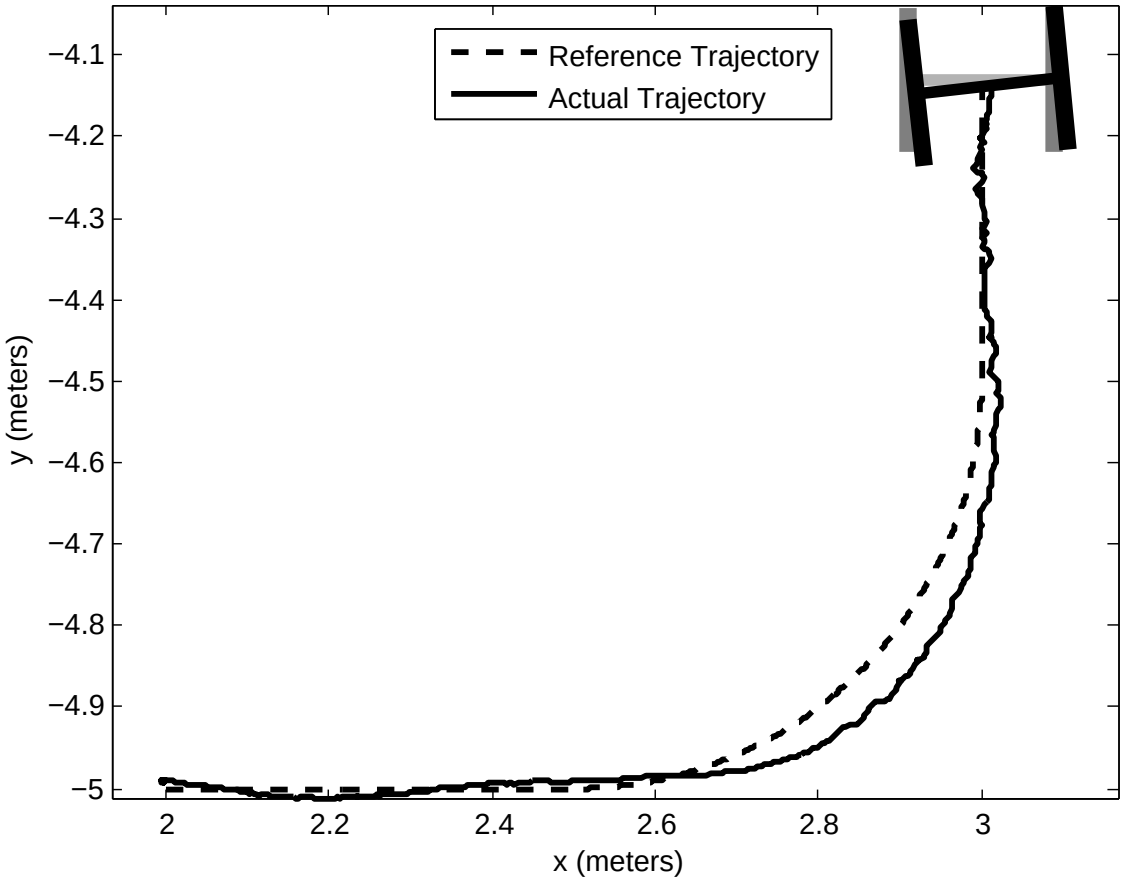

Figure 8.8. Three segment trajectory, final postures.

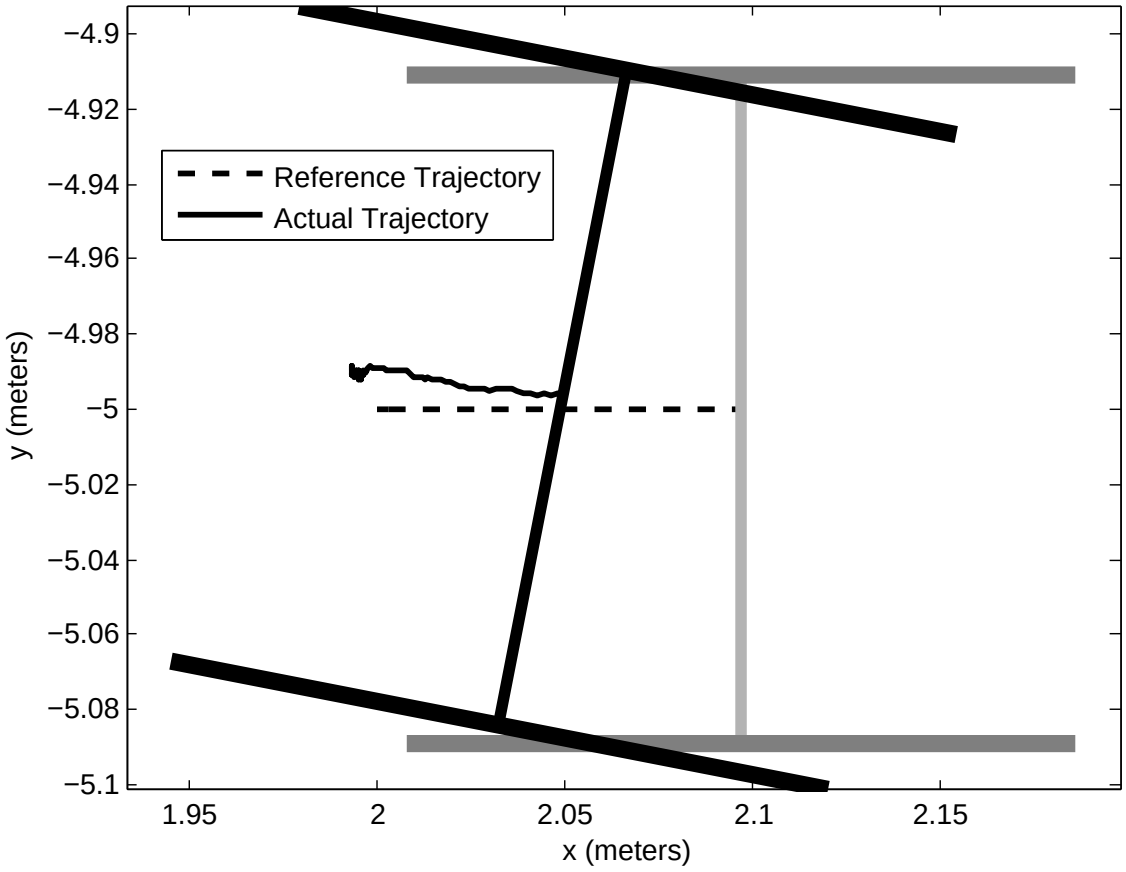

Figure 8.9. Three segment trajectory, initial convergence.

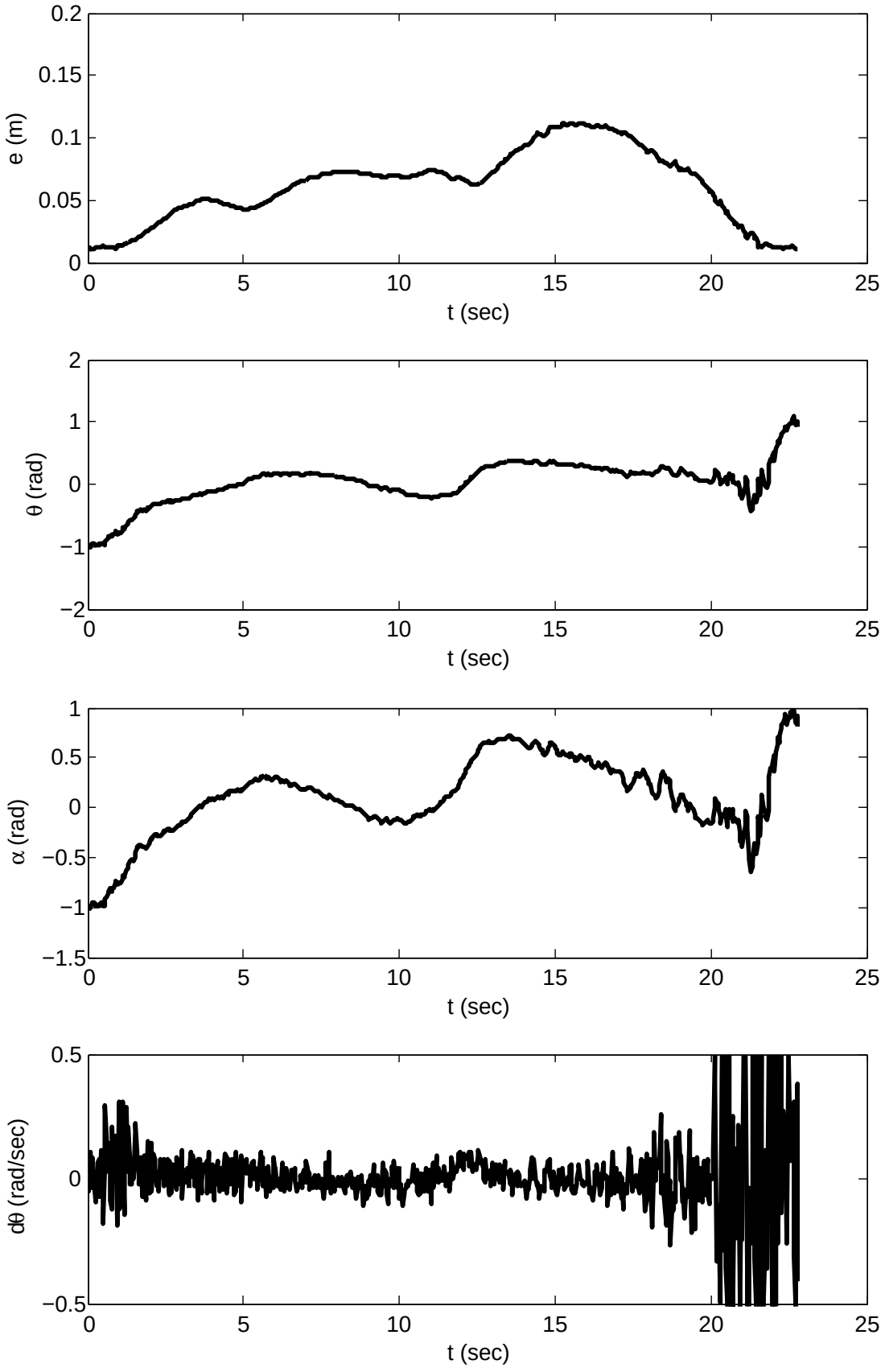

Figure 8.10. Three segment trajectory, Polar states.

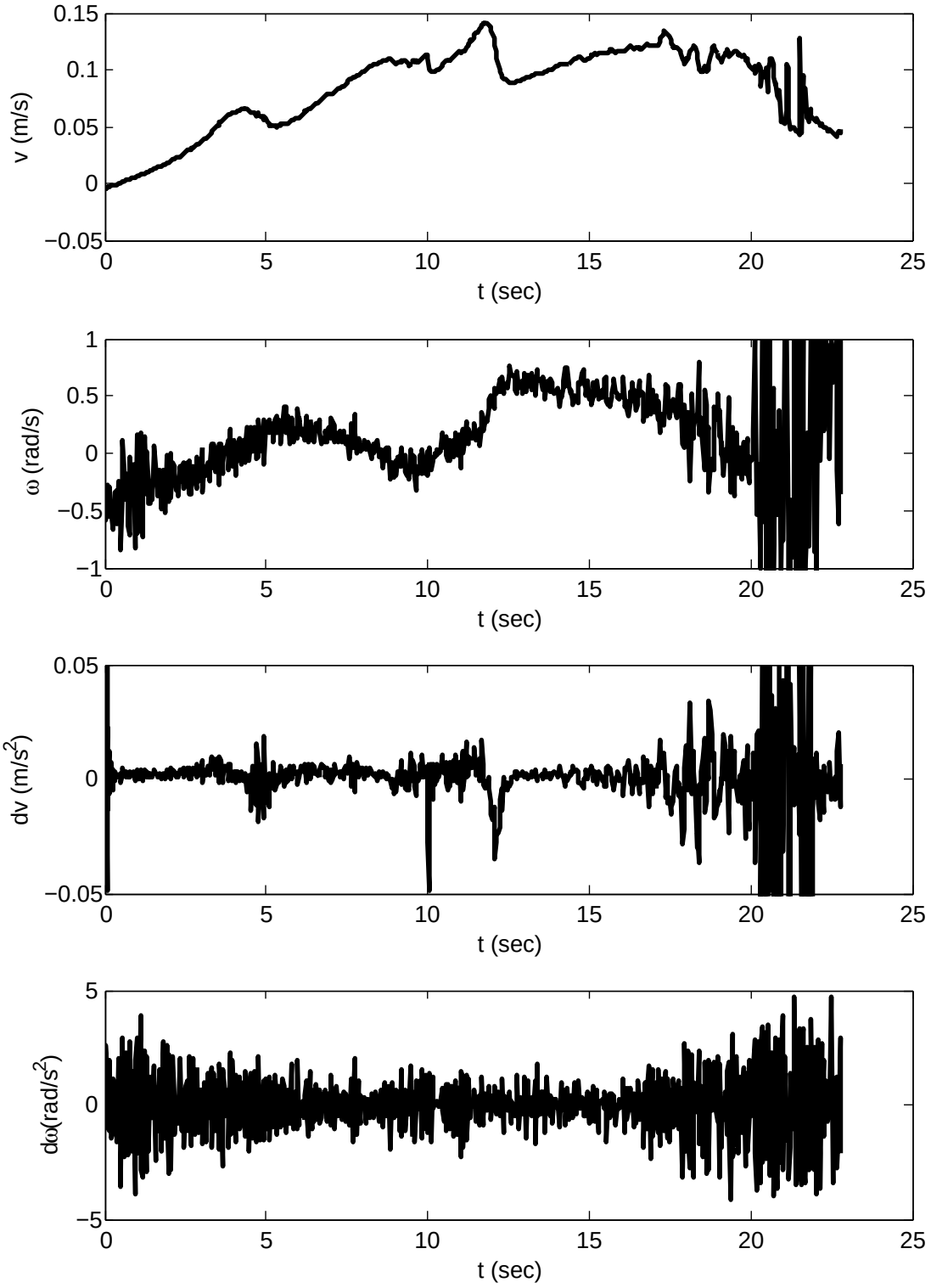

Figure 8.11. Three segment trajectory, controller velocity commands.

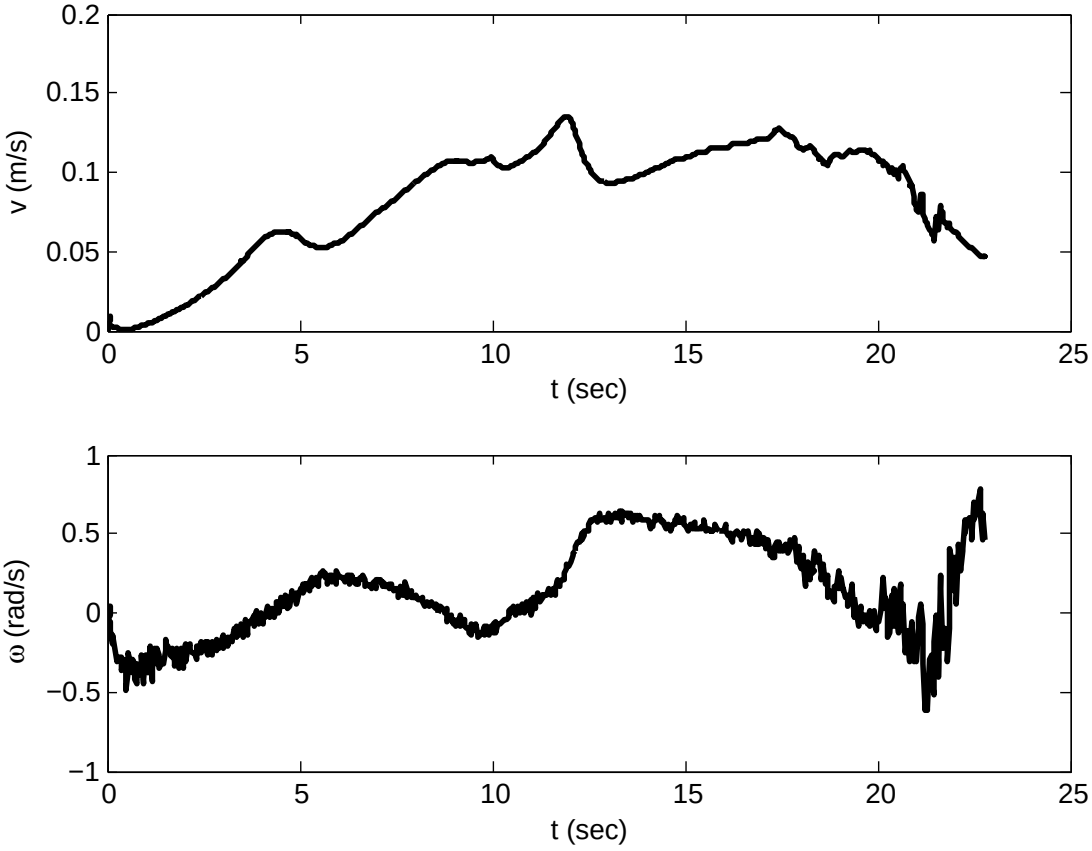

Figure 8.12. Three segment trajectory, dynamic extension output.

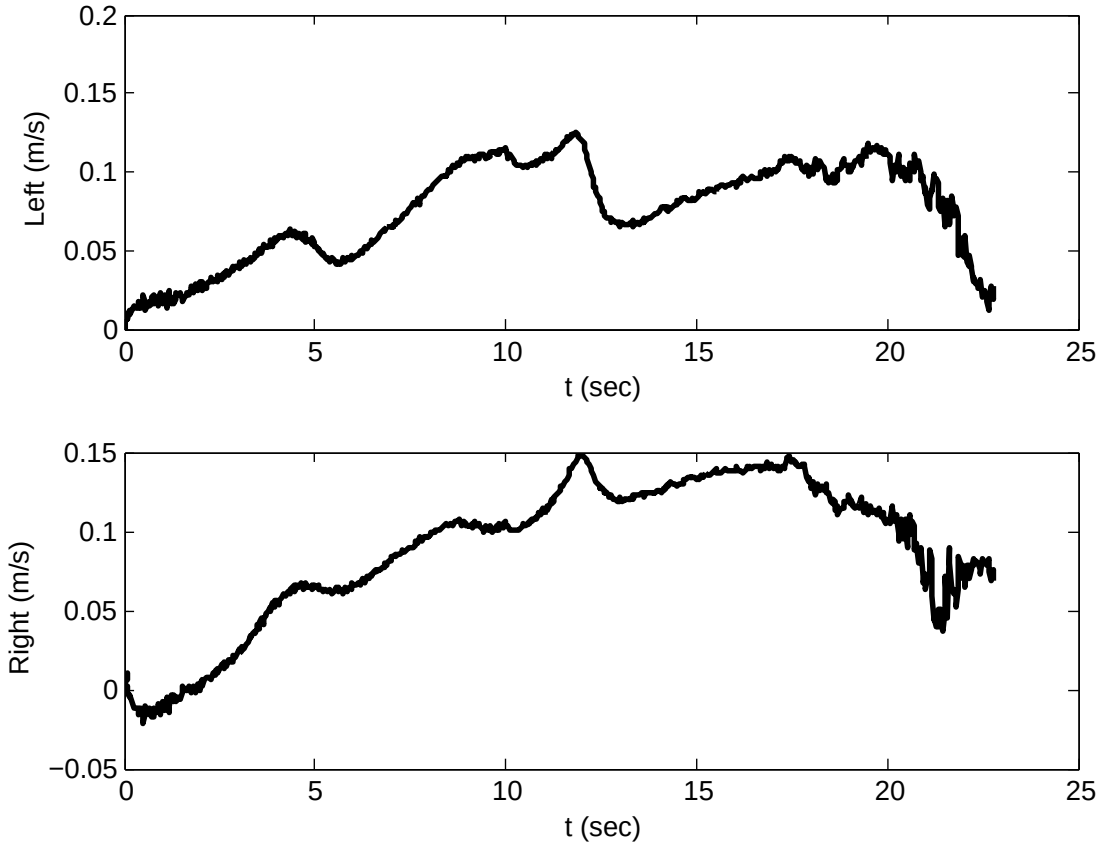

Figure 8.13. Three segment trajectory, wheel velocity commands.

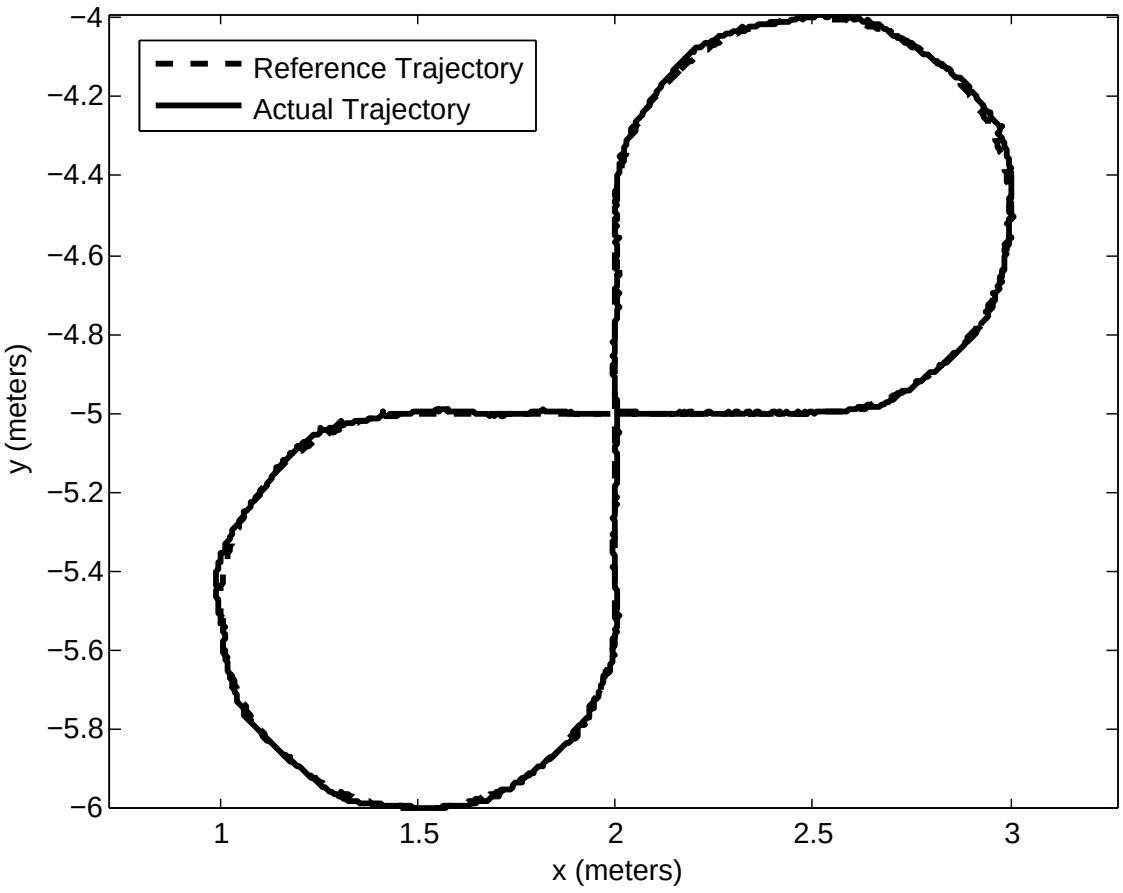

Figure 8.14. Trajectories for low speed experiment.

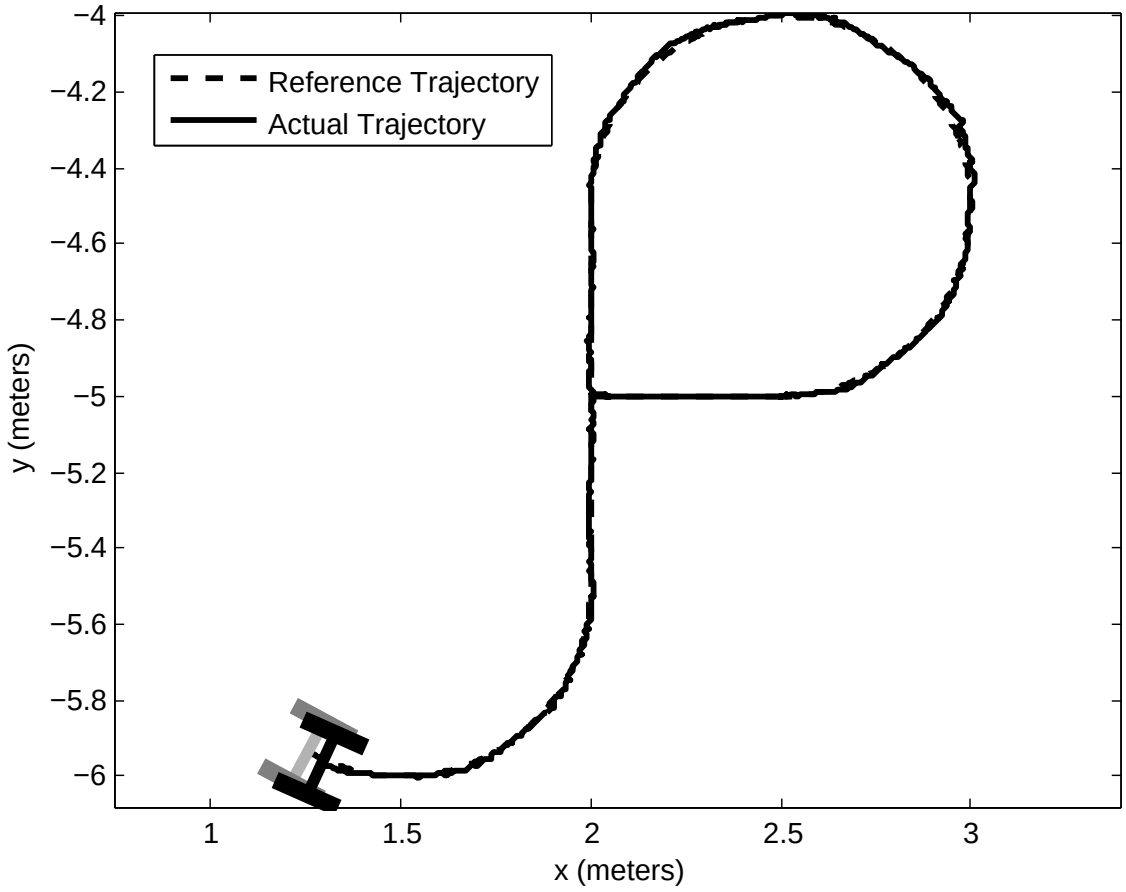

Figure 8.15. Trajectory and instantaneous robot position comparison for low speed experiment.

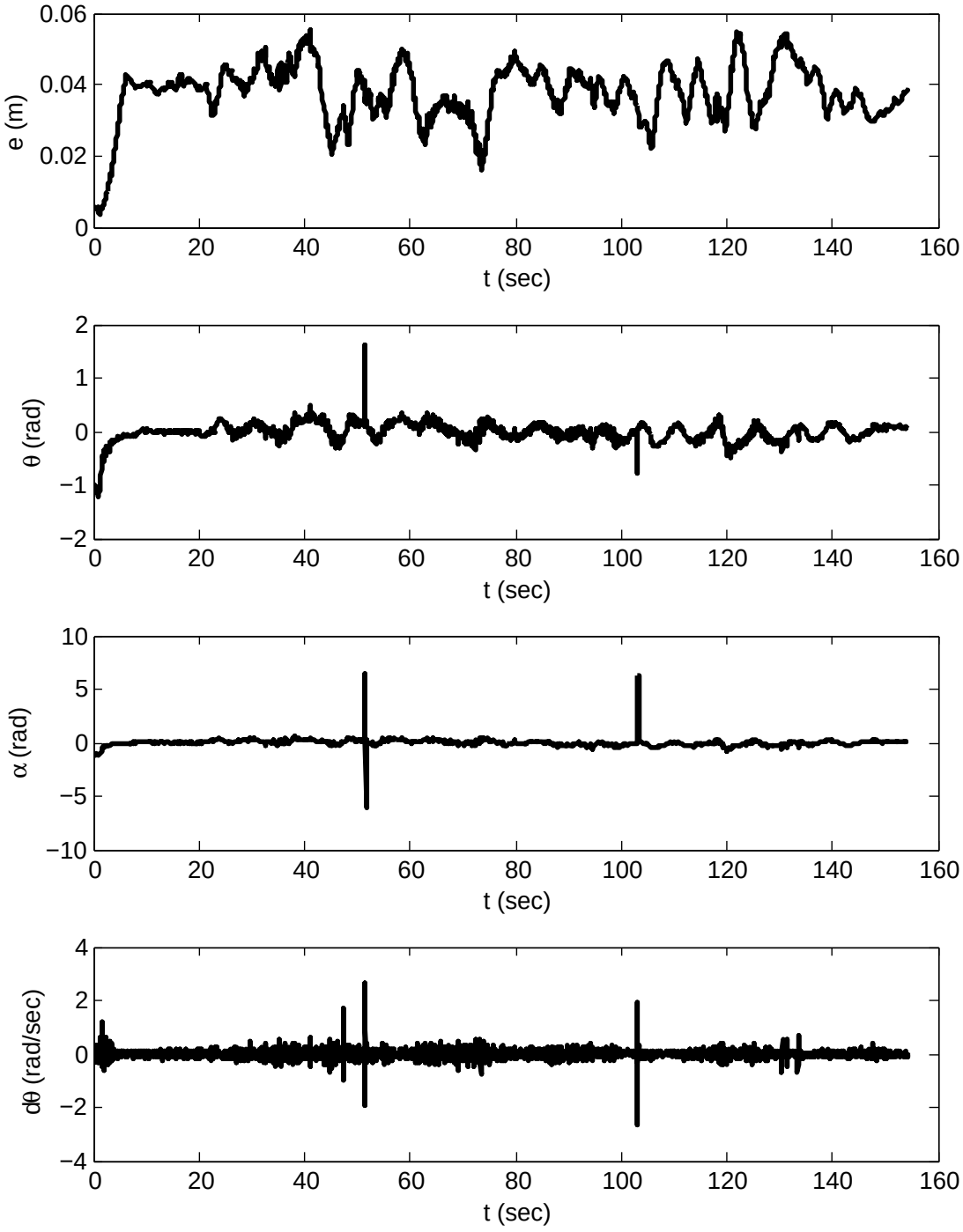

Figure 8.16. Polar states for low speed experiment.

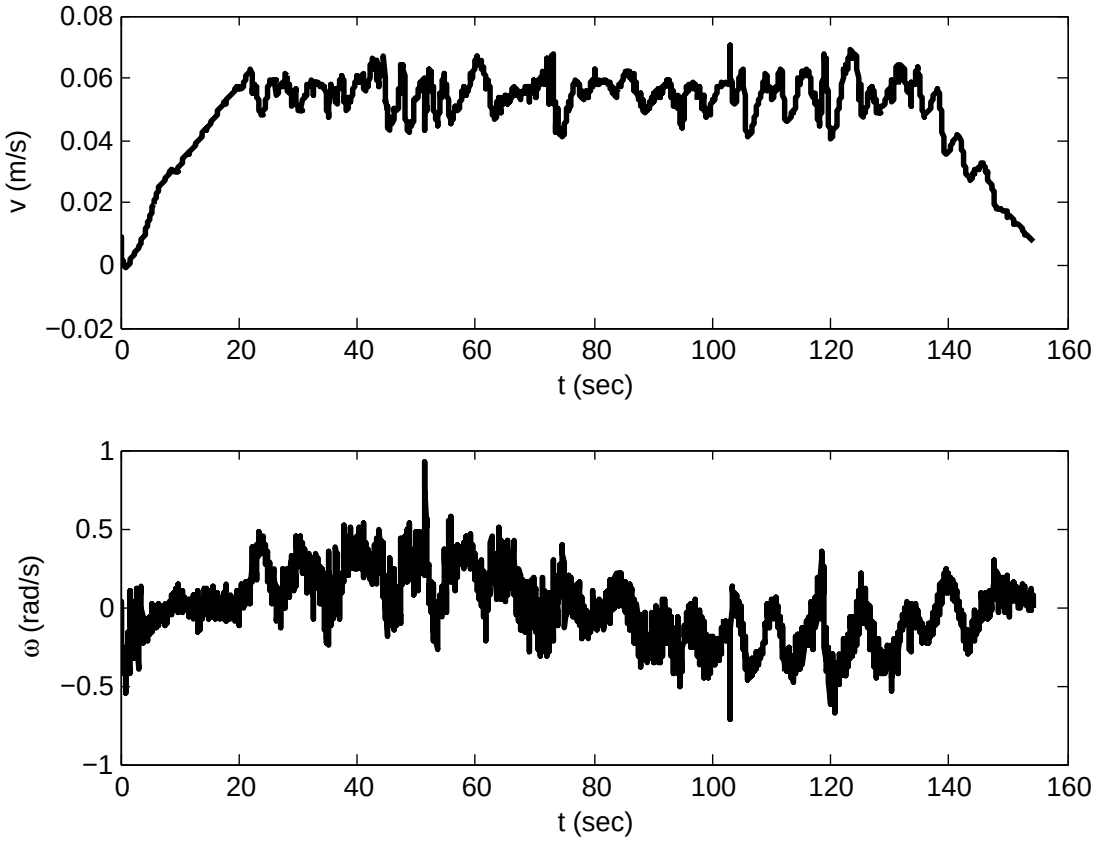

Figure 8.17. Dynamic extension output for low speed experiment.

8.1.5 Figure Eight Path

A second experiment using the figure eight path run in Subsection 8.1.4 is executed. The controller parameters for this experiment are given in Table 8.4. The resulting trajectory is presented in Figure 8.18. The begining and ending point of the trajectory is $[2.0 - 5.0]$, with a heading of $\phi = 0.0$. The maximum configured reference velocity is 0.1 meters per second, and the maximimum configured robot acceleration is 0.2 meters per second squared. There are curvature discontinuities in the path at $[2.5 - 5.0]$, $[2.0 - 4.5]$, $[2.0 - 5.5]$, and $[1.5 - 5.0]$. These discontinuities cause the tracking error, which is high in the instances where the robot transitions from a line segment to a curved segment. The curves used for the generated reference trajectory are constant radius circular arcs.

The Polar system states for this experiment are plotted in Figure 8.19. The maximum value of e is approximately 10 centimeters, which is within the maximum error tolerance

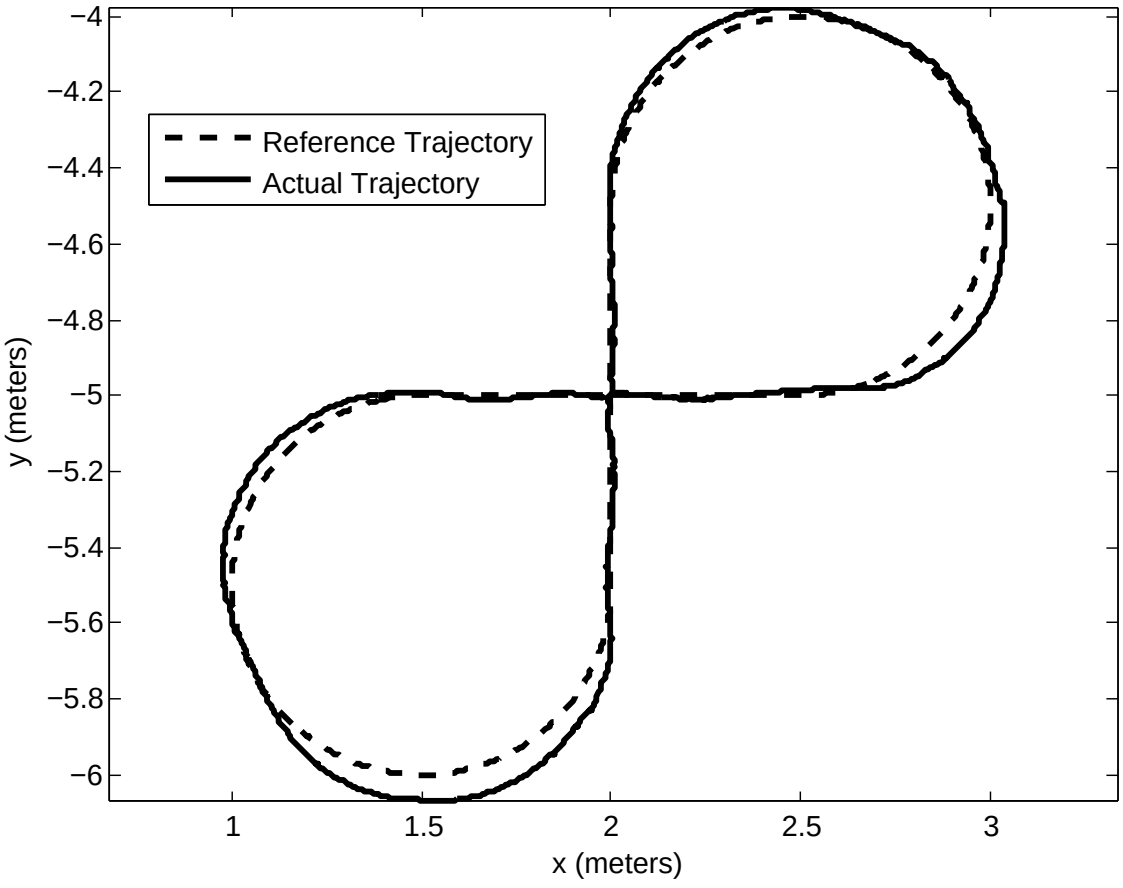

Figure 8.18. Trajectory for figure eight path experiment.

of 20 centimeters. The states θ and α remain within the interval $-\pi \rightarrow \pi$, as a result of the phase angle unwrapping function.

Figure 8.20 shows the controller velocity outputs v and ω, plus their respective time derivatives. The ω signal is noisy, because of noise present in the numerical differentiation of θ in the control law (4.42). There is also noise present in the numerical derivatives $\dot{v}$ and $\dot{\omega}$.

The velocity outputs of the dynamic extension are given in Figure 8.21. The velocity signals v and ω are smoother than the signals presented in Figure 8.20, as the dynamic extension acts as a low pass filter. At the time $t \approx 75$ seconds, a negative linear velocity is commanded as the robot overshoots the reference trajectory. At this point, the reference trajectory velocity is diminishing to zero.

The wheel velocities resulting from the velocity commands given in Figure 8.21 are plotted in Figure 8.22. These signals are sent directly to the robot for the low level wheel velocity controller to track. There is some jitter present, which is effectively filtered by the configured acceleration of 0.2 meters per second squared of the wheel velocity tracker.

The experiments run in this section demonstrate that the trajectory tracking controller performs as desired. There exist remaining issues, such as tracking error caused by noise and jitter. The maximum successful tracking reference velocity is also lower than desired. Overall, it is shown that repeatable results are possible, demonstrating stable trajectory tracking of complex curved paths at moderate velocities. Furthermore, it is established that the components of Emulab Mobile are capable of supporting kinematic state feedback for motion control over a multipurpose computer network system.

r	0.02 meters
ϵ	0.003 meters
k_1	0.85
k_2	0.3
k_v	3.0
k_c	3.0

Table 8.4. Controller parameters for figure eight path experiment.

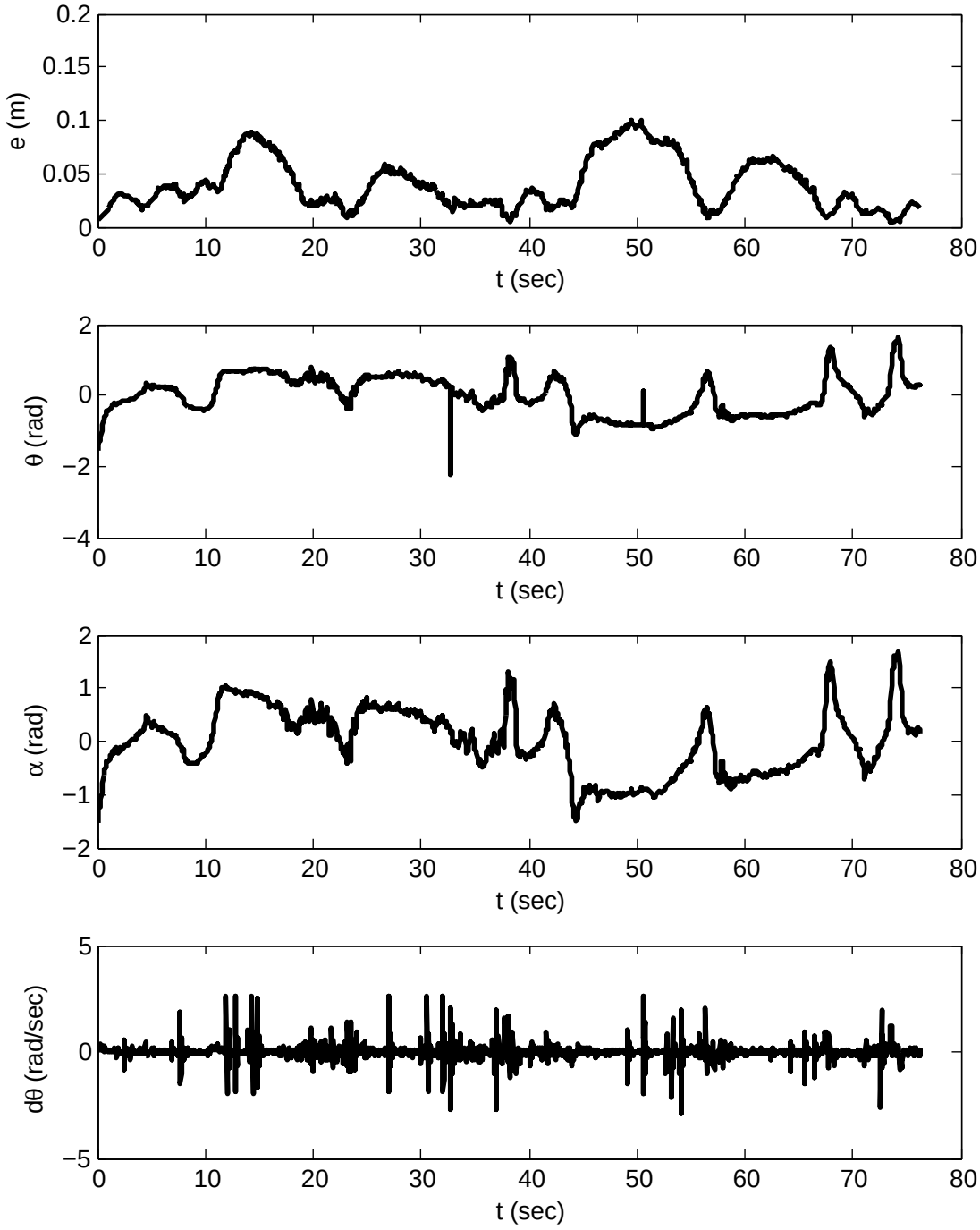

Figure 8.19. Polar system states for figure eight path experiment.

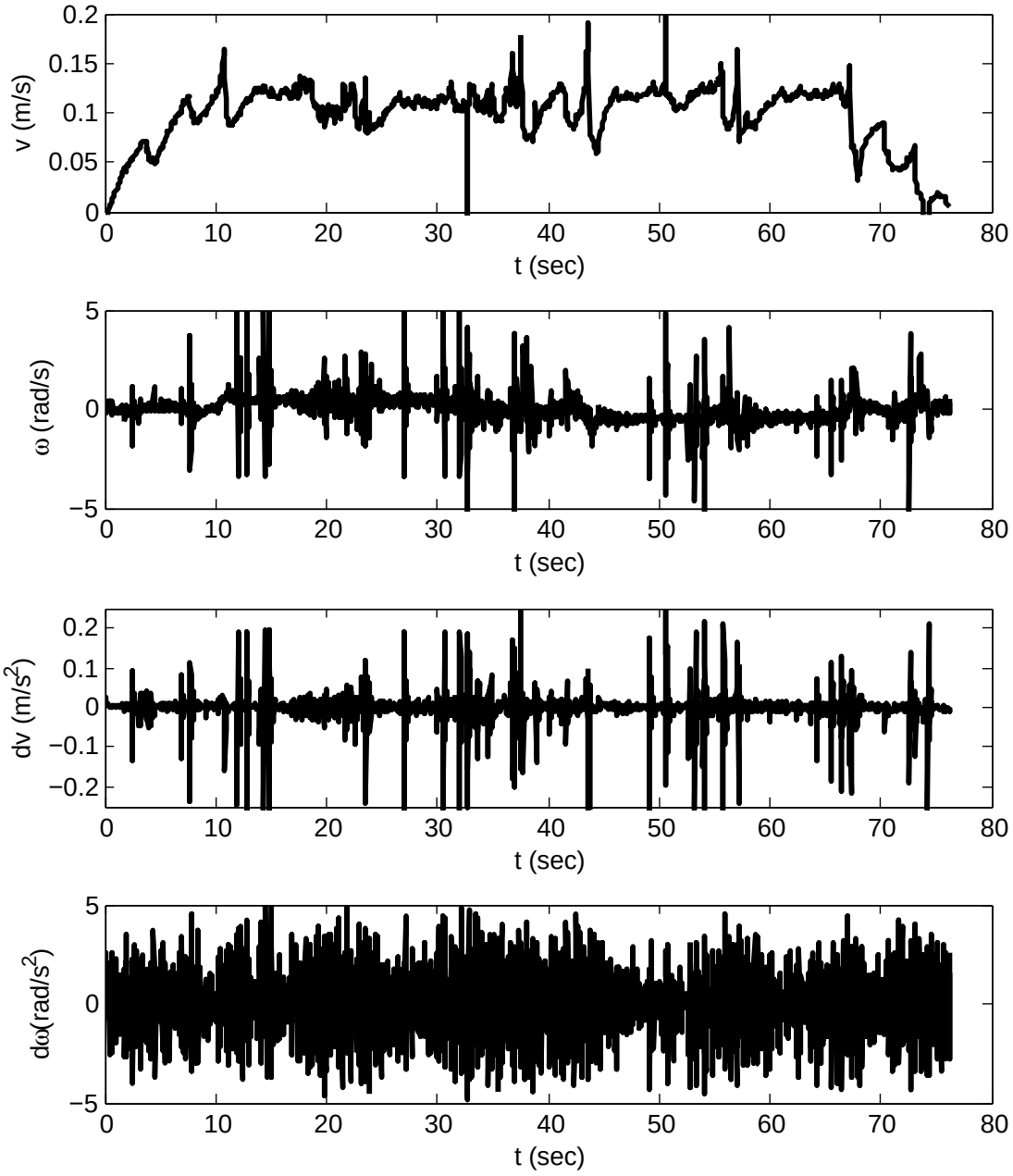

Figure 8.20. Controller velocity outputs for figure eight path experiment.

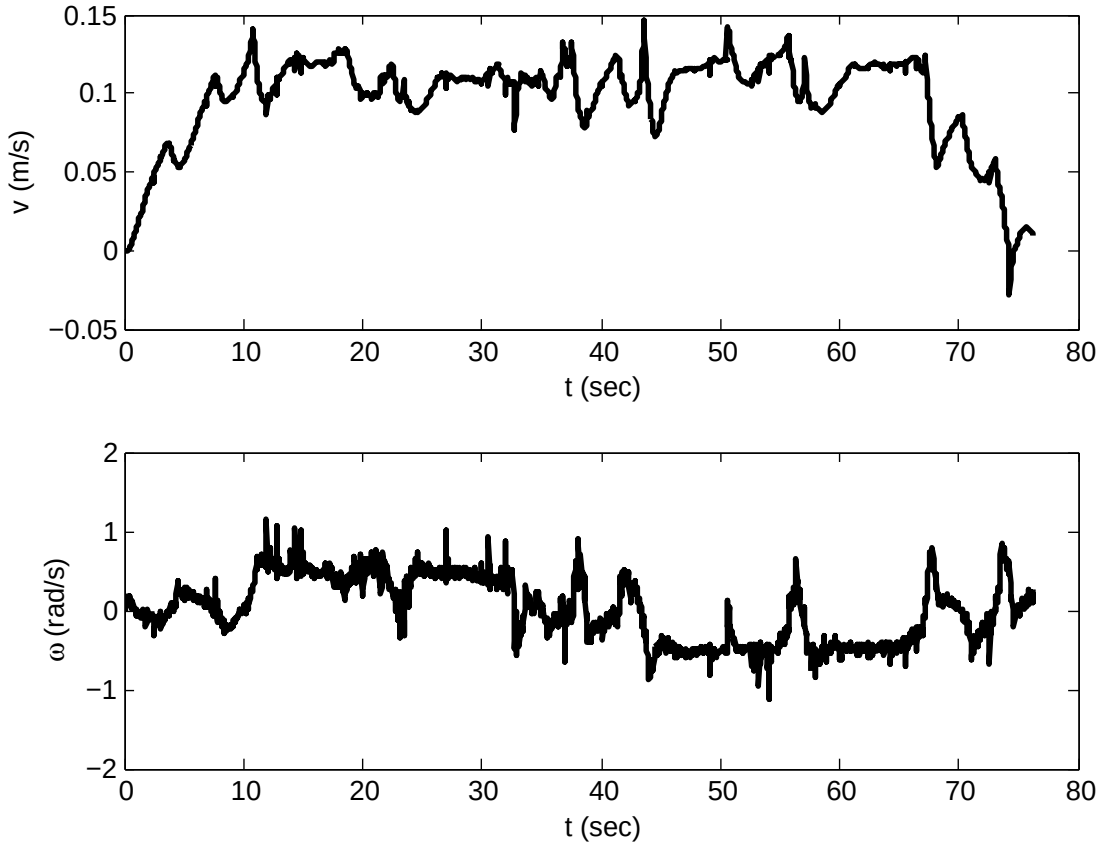

Figure 8.21. Dynamic extension velocity outputs for figure eight path experiment.

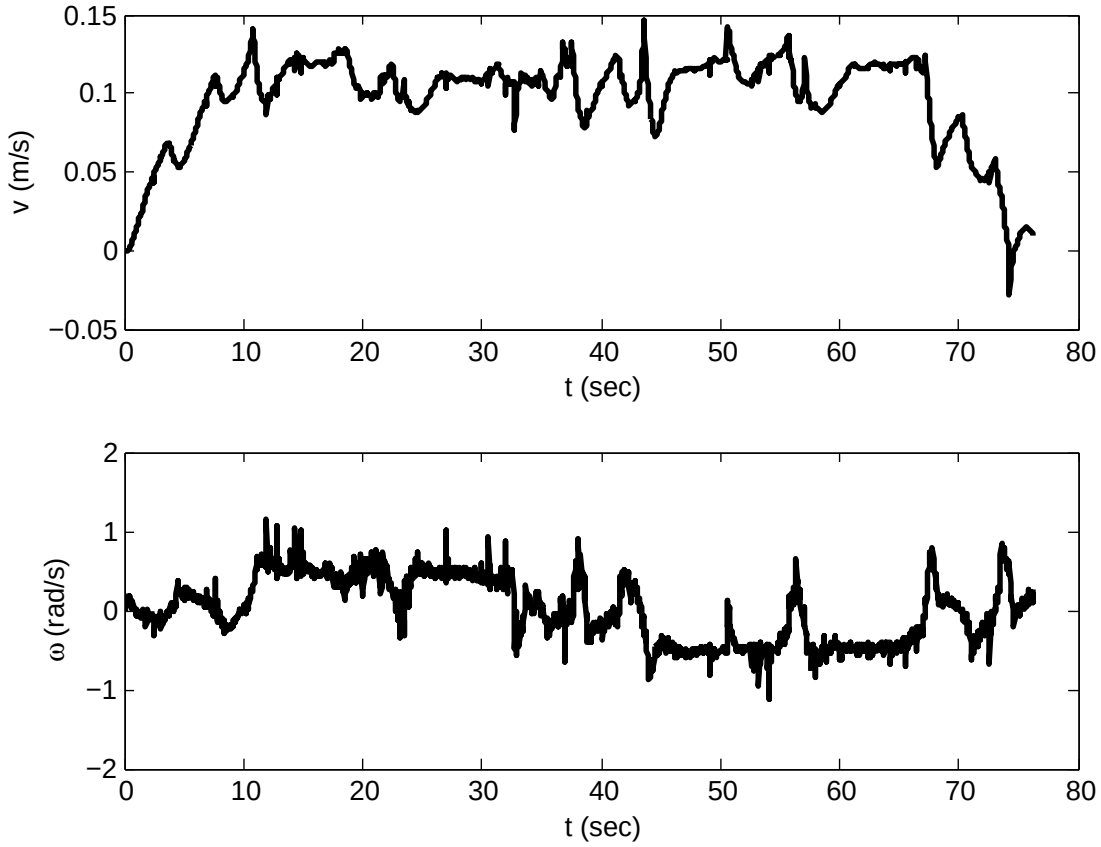

Figure 8.22. Wheel velocity commands for figure eight path experiment.

CHAPTER 9

DISCUSSION

The simulation data created in Chapter 6, and experimental data gathered in Chapter 8 are used to analyse the overall performance of the Emulab Mobile system, as related to the goals set for motion planning and control in Section 1.2. Despite the unexpected constraints that arose during the development of the system, the implementation of trajectory tracking control was successful. There remain performance issues, related to sampling and stability, but the system is tuned to reject these disturbances. The trajectory generator is not complete, with higher order curves not integrated in to the system. Overall, motion control with trajectory generation and tracking is successful on Emulab Mobile, and produces repeatable results.

The defining experiment for this research is illustrated by the trajectory given in Figure 8.18. The robot successfully tracks the provided trajectory, which is a figure eight shape with multiple curves. The trajectory has four discontinuities, which contribute to tracking error. The robot starts at $[2.0 - 5.0]$ with a heading of 0 radians, and ends the run at the same posture.

The initial posture is close to the initial point of the reference trajectory. The robot starts with an initial position error of approximately one centimeter. The controller used for motion control is designed to consider large initial errors. Initial errors are expected to be very low on Emulab Mobile, because robots are accurately positioned by the system prior to each movement.

As the robot proceeds in the global x direction through the first segment of the trajectory, tracking error is low. Once the first line to curve transition is encountered, the tracking error increases. The robot is not kinematically capable of tracking the trajectory at this point, since curvature must increase linearly. The tracking error increases again as the robot progresses around the first curve, and then oscillates back to the trajectory.

These oscillations are caused by instability introduced by variability in the sampling rate of the controller. As discussed in Section 7.4.6, the controller is not executed at

exactly 30 Hz. Variability is expected, but will become a problem by sending the system unstable as sampling frequency decreases. Noise and lag in the visual localization system further contribute to this problem.

The wheel acceleration limits of the robots may be increased, but this also contributes to stability problems. A lower acceleration limit acts as a low pass filter, and helps reject disturbances in the controller velocity commands. With lower acceleration limits, attainable velocities are limited by dimensional constraints in the workspace, and by the length of reference trajectories.

The problems with noise, lag, and sampling rate variability prevent the robots in Emulab Mobile from being operated at velocities above 0.1 meter per second while tracking C^0 continuous trajectories. The maximum error tolerance configured in Emulab Mobile is 20 centimeters. If tracking error becomes greater than this value, the motion controller will abort, and the trajectory execution will fail.

If the localization data sampling rate were to increase, coupled with a decrease in noise, less filtering would be required. This would benefit the stability of the system, and allow for higher velocities to be obtained. The localization system properties are considered as constraints in this research. As such, few actions can be taken to increase the maximum velocity performance of the robots significantly under discrete kinematic control.

The goal of successfully tracking trajectories with wheel velocities of 2.0 meters per second was not obtained. An accuracy of 1 centimeter for goal positioning was obtained at lower velocities. This is required for the accurate positioning of wireless experimentation hardware.

CHAPTER 10

CONCLUSION

In this chapter, final notes and comments about the components discussed in this research are presented. The experimental results discussed in Chapter 8 establish that the remote kinematic state feedback control of multiple robots simultaneously is feasible, and at controller sampling rates of 30 Hz. Stability constraints are satisfied, though velocity and acceleration performance is not as high as originally desired. To improve performance and reliability, future work is discussed.

In conclusion, a full motion planning and control system has been integrated into the Emulab Mobile wireless networking testbed system. Robots tracking complex trajectories under kinematic control are viable as couriers to carry wireless networking equipment to arbitrary locations within a workspace. Remote state feedback control is possible at low sampling frequencies, with inexpensive, commercially available robots and cameras.

Problems are encountered with system stability with the trajectory tracking controller. The robot actual performance falls short of the initially desired parameters. Higher order curves were not implemented in to the trajectory generator of Emulab Mobile, which caused tracking error to increase. In full, the project was successful, and the high level goal of delivering a working motion planning and control system to Emulab Mobile was completed.

10.1 Future Work

There are many improvements that can be made to the Emulab Mobile system. The trajectory generator needs to be upgraded to support more continuous curvature paths. The trajectory tracking controller needs more performance tuning, and more design work. The obstacle avoidance system needs development and integration. There are also user interface and software changes that are needed to increase the reliability and usability of the system.

10.1.1 Trajectory Generator

The trajectory generation system currently integrated in Emulab Mobile supports only line and circular arc segment trajectories. The inclusion of the spline based trajectory generator would improve robot performance, and benefit stability. The development of spline trajectories is complete, but implementation has not been attempted because of time constraints.

10.1.2 Motion Control

The trajectory tracking controller is still marginally stable under some circumstances, specifically when reference velocities are high, and the state feedback sampling rate is erratic. The current controller would benefit from the implementation of an Extended Kalman Filter for state estimation. With an EKF, not only would smoother and more accurate Cartesian state data be provided, but also velocity data, which would eliminate the need for the controller and dynamic extension to rely on noisy numerical differentiation for their velocity states.

A trajectory tracker of a different design may be necessary to improve the performance of the system. The current control law could be discretized and rigorously analyzed to create a digital control law better suited for the current system. A different controller design, such as a sliding mode controller may be integrated as a replacement to the current controller. The software infrastructure required to implement any sort of kinematic state feedback controller is already in place. A new controller may be substituted, requiring only minimal effort.

Major updates to the motion control system of Emulab Mobile can be performed. The addition of a dynamics based controller may significantly improve the performance of the robots. Moving the controller to a system offering real time performance would be highly beneficial. Emulab provides all the necessary components required to implement such a system.

10.1.3 Obstacle Avoidance

With support in VPPM for oriented obstacles, there are possibilities for future work involving the method. A bounding volume hierarchy could be used to speed computation, and model much more detailed workspaces. Greater control of rolloff functions, approach and departure zones, secondary exclusion zones, and field orientation affords more fine tuning of field properties. This also creates the preliminary support structure needed to

allow dynamic obstacles to be considered. The inclusion of support for dynamic (moving) obstacles would make VPPM better suited for use in solving multirobot coordination issues.

Due to the problems arising from interactions of multiple obstacles, VPPM may need to consider only a single obstacle at a time. Similar to some of the solutions for potential field methods, a controller or distance algorithm could be put in place to determine which obstacle or obstacles to consider as members of the field overlay. This could introduce more problems, especially oscillations and discontinuities.

VPPM could conceivably be combined with a trajectory generator, such as a probabilistic roadmap generator. While the trajectory generator creates a safe trajectory through a workspace or configuration space using geometric methods, VPPM could be used to allow dynamic or unforeseen obstacles to deflect a robot from the planned trajectory.

Combining VPPM with the waypoint based trajectory generator, could be used to add obstacle avoidance support to Emulab Mobile in the future. VPPM could be used to deflect user defined paths, routing rough paths around obstacles, and allowing robots to react to transient obstacles that may appear in the workspace.

To solve the problem concerning trajectories entering the obstacle exclusion zone on approach, the goal attractor field could be switched to repulse in order to drive the trajectory to a safe distance from the obstacle. To minimize the impact of a discontinuity, this switch could be accomplished with a saturation function, similar to methods commonly employed in sliding mode controller design. The obstacle field overlay itself could react to the current posture by changing strength, shape, or orientation. We could use these efforts to make VPPM more adaptive, while using all of its existing attributes.

10.1.4 Implementation

The software running Emulab Mobile may be improved for usability and reliability. The current user interface does not support the segmented waypoint trajectory generators, and there is little accommodation for state feedback control outside of RMCD. A localization system with cameras capable of faster framerates would increase the sampling rate of the controller and improve performance and stability. Robots with better performance characteristics, such as maximum velocity and acceleration, would improve the overall performance of the system.

REFERENCES

[1] D. Johnson, T. Stack, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile emulab: A robotic wireless and sensor network testbed," in *Proc. of the 25th IEEE Conference on Computer Communicati ons (INFOCOM 2006)*, Barcelona, Spain, Apr. 2006.

[2] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002, pp. 255 – 270.

[3] J.-P. Laumond, P. Jacobs, M. Taix, and R. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 577 – 93, Oct. 1994.

[4] B. Barsky and T. DeRose, "Geometric continuity of parametric curves: three equivalent characterizations," *IEEE Computer Graphics and Applications*, vol. 9, no. 6, pp. 60 – 9, Nov. 1989.

[5] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 1025 – 35, Dec. 2004.

[6] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila, "Primitives for smoothing mobile robot trajectories," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 441 – 448, 1995.

[7] D. Walton and D. Meek, "A controlled clothoid spline," *Computers & Graphics*, vol. 29, no. 3, pp. 353 – 63, June 2005.

[8] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 583 – 601, July 2003.

[9] Y. Kanayama and B. Hartman, "Smooth local path planning for autonomous vehicles," *Proceedings. 1989 IEEE International Conference on Robotics and Automation (Cat. No.89CH2750-8)*, pp. 1265 – 70, 1989.

[10] J. Reuter, "Mobile robots trajectories with continuously differentiable curvature: an optimal control approach," *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, vol. vol.1, pp. 38 – 43, 1998.

[11] R. Fierro and F. Lewis, "Control of a nonholonomic mobile robot: backstepping kinematics into dynamics," *Proceedings of the IEEE Conference on Decision and Control*, vol. 4, pp. 3805 – 3810, 1995.

[12] I. Kolmanovsky and N. McClamroch, "Developments in nonholonomic control problems," *IEEE Control Systems Magazine*, vol. 15, no. 6, pp. 20 – 36, Dec. 1995.

[13] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino, "Closed loop steering of unicycle-like vehicles via lyapunov techniques," *IEEE Robotics & Automation Magazine*, vol. 2, no. 1, pp. 27 – 35, 1995.

[14] Y. Kim and M. A. Minor, "Path manifold based kinematic control of wheeled mobile robot considering physical constraints," *submitted to IEEE Trans. Robotics*, 2006.

[15] G. Indiveri, "Kinematic time-invariant control of a 2d nonholonomic vehicle," *Proceedings of the IEEE Conference on Decision and Control*, vol. 3, pp. 2112 – 2117, 1999.

[16] V. Utkin, S. Drakunov, H. Hashimoto, and F. Harashima, "Robot path obstacle avoidance control via sliding mode approach," *Proceedings IROS '91. IEEE/RSJ International Workshop on Intelligent Robots and Systems '91. Intelligence for Mechanical Systems (Cat. No.91TH0375-6)*, pp. 1287 – 90, 1991.

[17] Y. Ma, J. Kosecka, and S. S. Sastry, "Vision guided navigation for a nonholonomic mobile robot," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 521 – 536, 1999.

[18] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," *Proceedings 1990 IEEE International Conference on Robotics and Automation (Cat. No.90CH2876-1)*, pp. 384 – 9, 1990.

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots." *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90 – 98, 1986.

[20] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," *Proceedings. 1991 IEEE International Conference on Robotics and Automation (Cat. No.91CH2969-4)*, pp. 1398 – 404, 1991.

[21] K. A. McIsaac, J. Ren, and X. Huang, "Modifed newton's method applied to potential field navigation," *Proceedings of the IEEE Conference on Decision and Control*, vol. 6, pp. 5873 – 5878, 2003.

[22] C. I. Connolly, J. B. Burns, and R. Weiss, "Path planning using laplace's equation," pp. 2102 – 2106, 1990.

[23] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224 – 41, 1992.

[24] J. Guldner and V. Utkin, "Sliding mode control for an obstacle avoidance strategy based on an harmonic potential field," *Proceedings of the 32nd IEEE Conference on Decision and Control (Cat. No.93CH3307-6)*, vol. vol.1, pp. 424 – 9, 1993.

[25] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," *IEEE International Conference on Intelligent Robots and Systems*, vol. 4, pp. 3546 – 3551, 2003.

[26] "Garcia robots from acroname, inc." http://www.acroname.com/garcia/garcia.html.