PATH PLANNING AND CONTROL OF A NONHOLONOMIC AUTONOMOUS ROBOTIC SYSTEM FOR DOCKING

By

CHADWICK ALLEN SYLVESTER

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

This document is dedicated to the person whom my life is indebted to, Jesus the Christ.

ACKNOWLEDGMENTS

The author would like to acknowledge his committee members, Dr. Gloria J. Wiens (chair), Dr. Norman Fitz-Coy (cochair), and Dr. Andrew Kurdila for their exhortation and motivation in driving this research in its attention to detail.

This work has been funded in part through NASA Cooperative Agreement NCC3-994, the Institute for Future Space Transport University Research, Engineering and Technology Institute (URETI).

The author also acknowledges Tracy McSheery and Phasespace Inc. for their interest and contributions of the position measurement equipment.

The author is greatly indebted to the friendship and competition found in the members of the SAMM (Space, Automation, and Manufacturing Mechanisms) Laboratory: Kaveh Albekord, Jessica Bronson, Jean-Philippe Clerc, Young Hoon Oh, Umesh Tol, and Adam Watkins.

The author would like to thank his family for their support and love: his parents, Timothy Wayne and Beverly Ruth Sylvester, his brothers, Matthew James and Christopher Brent Sylvester, and his selfless wife, Cheryl Joy Sylvester, who compassionately assisted his endeavors throughout this research. Further appreciation is extended to the author's church family, Fellowship Baptist Church, for their continual encouragement.

TABLE OF CONTENTS

Page

APPENDIX

LIST OF TABLES

LIST OF FIGURES

Figure page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

PATH PLANNING AND CONTROL OF A NONHOLONOMIC AUTONOMOUS ROBOTIC SYSTEM FOR DOCKING

By

Chadwick Allen Sylvester

December 2003

Chair: Gloria J. Wiens
Major Department: Mechanical and Aerospace Engineering

Autonomous maneuvering can be a difficult and broad issue for multiple robots performing various tasks. In the coordination control of multiple robots, compatible trajectories for each robot are essential. This is especially critical for nonholonomic robots performing collaborative tasks. To ensure proper movements performed by the robots, trajectories in time must first be achieved to guarantee efficient, non-impeding traversals. The applications where this is pertinent are transportation, surveying, formation and following. Position feedback and path planning are two major areas of study that have developed from these endeavors.

In this thesis, an architecture is presented and implemented that uses classical control methods to track the position of two robots in a workspace. The goal of their maneuvers is to have the parallel-steered robots traverse a predetermined trajectory and result in a configuration that allows for front-to-front docking. A non-model based approach is applied in the development of a path planning algorithm for tracking control

subject to physical constraints of the robot's maneuverability. A framework for creating a trajectory via five constraints applied to a fourth order polynomial is presented and implemented. The predetermined trajectory is obtained using a simplified polynomial bisection search technique for finding "optimal" polynomial coefficients. The polynomial based trajectory also ensures that the minimum turning radius of each vehicle is not violated for any part on the trajectory. The controller incorporates the speed and steering to achieve the desired position and heading of the robots as well as ensure that each vehicle traverses the prescribed trajectory to a docking location.

Using an off-board 3D camera system and robots equipped with LEDs, this algorithm is implemented in a tracking controller for two parallel-steered robots required to maneuver and dock with one another. Both the speed and steering controls require delicate tuning of gains for accurate responses. Various experiments are performed while modifying the initial poses of the vehicles. Tuning methods are used as opposed to a model-based approach because the inaccuracies of the steering mechanism and drive gearbox would negate the advantages achieved by the dynamical model obtained.

The outcome of this research is the fundamental path planning architecture for a hierarchical controller in which the robots' pose is sent to the collective intelligence of the team where the decisions are generated to plan trajectories and control the robots in coordinated tandem maneuvers. The corresponding position measurement system is an overhead, off board system, which allows for collection of global information about the positions of the robots and their goals.

CHAPTER 1
INTRODUCTION

Much research has been performed in the area of wheeled mobile robots. Dynamics, controls, and structural analysis have been extensively developed for these machines. With the advancement of the dynamics and controls have come many branches of study such as path planning, sensory limitations, and cooperative teams. Each of which will be briefly discussed in this research. This work will present more specifically the integration of these studies as applied to multi-agent cooperative robotic systems.

1.1 Applications of Multi-Agent Robots

Much work has been done in the area of autonomous multi-agent wheeled mobile robots including many different types of multi-agent projects each with its own fields of specific research. The following is a short list of some of these topics. Each will be discussed with a brief description of their application to this thesis.

- Formation and Following
- Transportation
- Mine Deactivation
- Cleaning

1.1.1 Formation and Following

Several aspects of formation and leader following studies have been developed. Similar to other areas of multi-robotic system, these algorithms are best maintained when

information and decisions are developed both on the team level and on an individual level.

Oak Ridge National Laboratories and the University of Podova [Car02] described such a method where each robot follows a designated robot. As an obstacle enters one robot's path, in order to account for this hindrance all of the robots stop and wait a predetermined time for the obstacle to move. If the obstruction does not leave the robot's planned path, all of the robots enter a recovery mode where the robots move slowly as the hindered robot maneuvers around the obstacle and reenters the formation. Figure 1.1 shows an example of a leader following routine.

Figure 1.1. Formation and following performed by Oak Ridge National Laboratory

This algorithm seems to be good for ground-based vehicles; however, there are many other applications for formations such as aircraft. Figure 1.2 illustrates the leader following configuration with high-speed aircraft as used by NASA Dryden Research Center. The waiting algorithm would work for this scenario because the obstacles encountered by aircraft are typically stationary while the vehicle itself cannot be. These machines encounter a set of entirely different problems including precision, physical limitations, and energy constraints that make formation flying a difficult process.

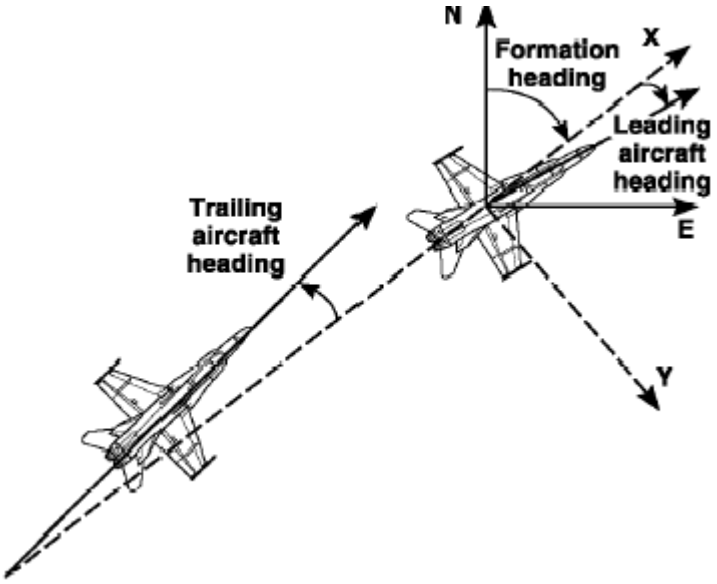

Figure 1.2. Formation and following at NASA Dryden Research Center

As opposed to the formations where a robot is only focused on the robot it follows, the MAGICC Laboratory at Brigham Young University [Law00] suggests an algorithm that allows each robot to incorporate the knowledge of the location of the robot that is leading it and the robot(s) following it. This allows for the team of omnidirectional (Hilare-type) robots to consider individuals that are slower or are attempting to recover the formation (i.e. after avoiding an obstacle). They have further developed this knowledge from a non-linear state of equations to a feedback-linearized system for translations, rotations, and contractions/expansions.

The Universita di Roma Tor Vergata [Gen00] has further developed this formation problem with the additional constraint of shortest path optimization. By adding the lengths of the predetermined paths of all of the robots to get from their initial positions to their final positions to complete the formation, an optimal solution for the centralized algorithm can be computed. This approach is quite significant for the formation of vehicles. This algorithm can be adapted to energy and time constraints as well; however; this centralized approach can be extremely computationally expensive

1.1.2 Transportation

Another task-oriented algorithm improved by a team of multiple robots is that of transportation of objects that are either too heavy or awkward for a single robot to carry. Some of the basic algorithms start by pushing an object across the floor to a goal destination. Others configure a pattern around the object based on the distribution of weight to allow maximized lifting capabilities among the robots.

Miyata et al. [Miy00] from the Mobile Robots and Advance Robots Laboratories at the University of Tokyo have landscaped an algorithm that allows a team of robots to handle an object while others in the team search around for obstacles, remove obstacles, or measure landmarks. Though each of the robots could do any one of the tasks, any single robot consumes fewer resources such as computing power, time, and energy when teams are formed. Furthermore, they have laid groundwork for real-time task assignment.

Pereira et al. [Per02] of the VERlab of the Universidade Federal de Minas Gerais developed an algorithm where the robots change leader follower status during an operation of maneuvering around an obstacle. Figure 1.3 shows the movement of the robots with respect to the object they are carrying. The robots can measure both the linear and angular movements of the box. Figure 1.4 shows the path of the robots as they switch leader follower status and move toward the goal.

While this methodology solves several problems, it is still limited to a plane of obstacles. Yamashita et al. [Yam00] have created an architecture that will allow planar robots to move under, around, and in between obstructions. This algorithm presented in Figure 1.5 incorporates the global measurements of the position of the robots, the position of the object, and the posture of the object. While knowing the obstacles in the

environment, this centralized approach determines the capabilities and an efficient use of the robots in the workspace.

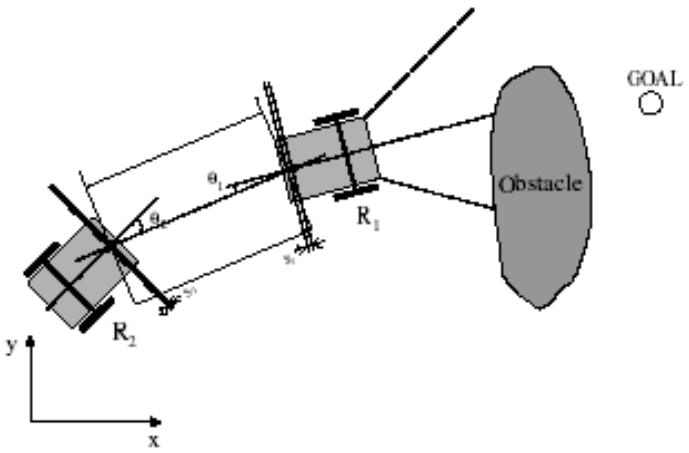

Figure 1.3. Two robots carry an object as they approach an obstacle where linear and angular movements can be detected

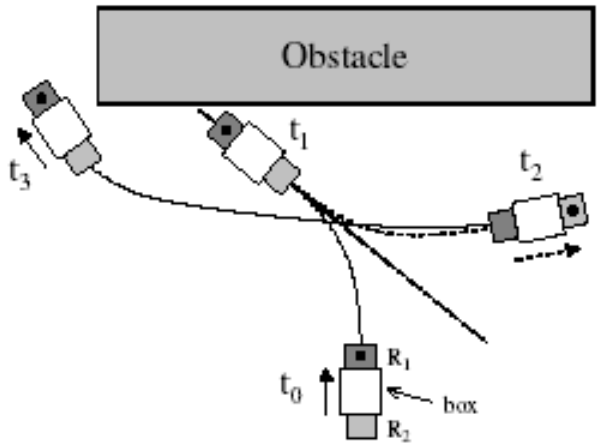

Figure 1.4. Two robots carry an object as they approach and attempt to maneuver around the obstacle

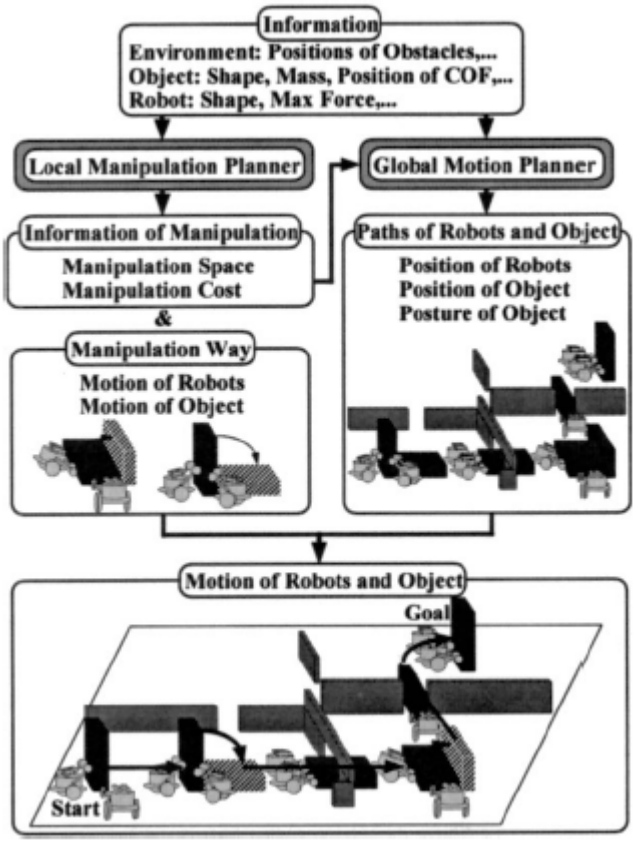

Figure 1.5. An algorithm for 3D movements for transporting an object around obstacle with multiple robots as presented by the University of Tokyo

1.1.3 Cleaning

Cleaning can range from the inspection of a vehicle or field to physical removal of unwanted material in an area. Whatever the application, quality is a key issue. Corporate Technology and the Institute fur Informatik teamed together on a project for such an application [Jag02]. Their approach is decentralized with communication between the robots only available within a given range. The robots, if close enough, share information about what they know to have already been cleaned and what is presently being cleaned. With this information, the robots concentrate on areas where the clean status of an area is unknown to them, and therefore are considered unclean. This algorithm allows the workspace to be cleaned much quicker than a single robot. Because

direct communication is not always available between robots, some parts may be cleaned more than once. Figure 1.6 illustrates an unknown territory being cleaned by three robots. The black areas are clean regions. The robots are presently cleaning the dark gray areas. The white areas area are undiscovered. The overlapping circles imply communication is available.

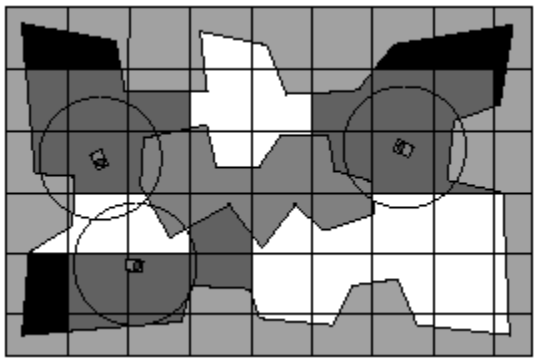

Figure 1.6. A decentralized approach to cleaning with sparse communication

1.2 Problem Statement

This thesis will develop an autonomous path/trajectory planning team of two multi-tasking robots in a controlled environment that share position data en route to docking with each other or individually to a stationary object while constrained by physical space, turning radii, and power limitations.

1.3 Fundamental Issues

Robots are made autonomous by their behaviors. There is nothing particularly interesting about a robot that can travel in a circle until the robot has decided for itself that it needs to travel in a circle. Assessments of goals and their environment further develop these machines into “intelligent” robots allowing them to choose one command or trajectory over another. Decisions can be made from individually gathered data or by information they might have received from a team member. Within the study of multi-

agent robotics there are certain fundamental issues that must be considered before any sort of theoretical approach can be made. The following is a conservative list of the most important problems to discuss.

- Knowledge of Coordinates
- Communication
- Parallel Steering
- Path Planning
- Docking

1.3.1 Knowledge of Coordinates

Two methods for position feedback are on board and off board. Image capturing for on board requires that the camera be mounted to the robot. This does not require processing on board; the information may be sent to another computer with faster processing capabilities. Off board feedback incorporates a camera that is not mounted on the vehicle needing the position feedback, but sees that robot and its target. An overhead position for downward facing cameras is popular for tracking planar robots (e.g., Global Positioning Systems (GPS)).

Saedan et al. [Sae01] investigate the use of a camera with object finding for the precision control of an industrial robot. By enabling the processor with a view of the object in image space, scaling and coordinate transformation can used to impose pattern recognition for visual feedback. Figure 1.7 gives an elementary depiction of the use of a camera for feedback. The pinhole camera on the end effector provides the processor with a two dimensional image of the object. Image processing is then performed with target pose estimation. Saedan and Ang further apply classical control methods for trajectory planning of a serial robot.

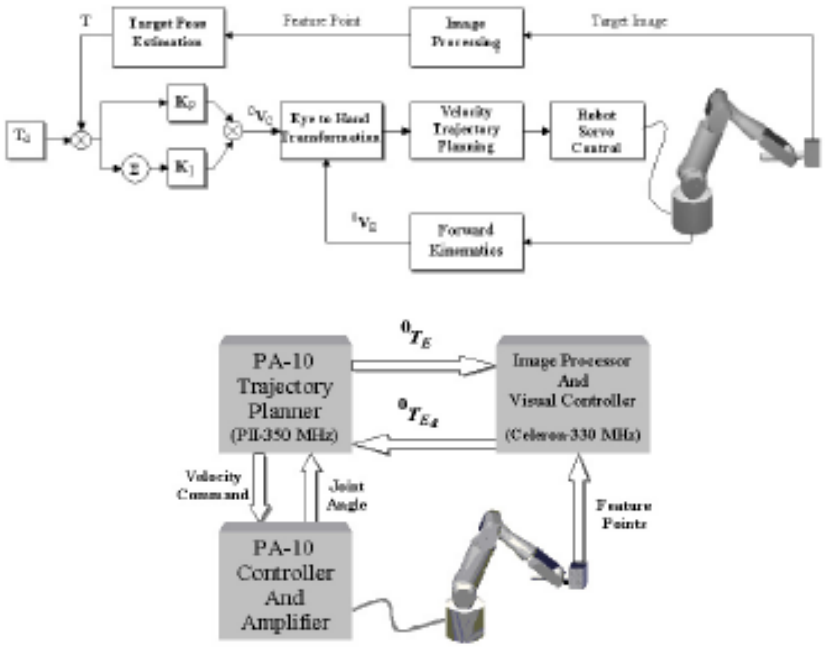

Figure 1.7. Control of an industrial serial robot using visual feedback

Das et al. [Das01] of the General Robotics Automation, Sensing and Perception (GRASP) Laboratory have implemented an Omnicam for vision feedback to control a car-like mobile robot (figure1.8). The camera provides a 360^o field of view, which allows for obstacle sensing. The camera provides color images via a 2.4 GHz wireless transmitter to a PC workstation, which controls the robots. Wall following, obstacle avoidance, and leader following are employed. The wall following and obstacle avoidance are done with simple error feedback with respect to the wall or obstacle. The leader follower task is slightly more complex with the follower robot requiring reliable knowledge of the linear and angular velocities of the leader robot. With this on board camera, relative position can be obtained quite easily, but integrated landmark recognition for a coordinate knowledge can be much more difficult.

In addition to object detection and 360^o vision feedback, shape recognition is a viable method for position feedback [Lei98]. Polynomial fitting is used to test the shapes

in the image from the on board camera. If objects in images can be simplified into shapes, vision feedback for landmark could be manipulated with a few more criteria to determine size and distance. Figure 1.9 shows an example of the shape recognition.

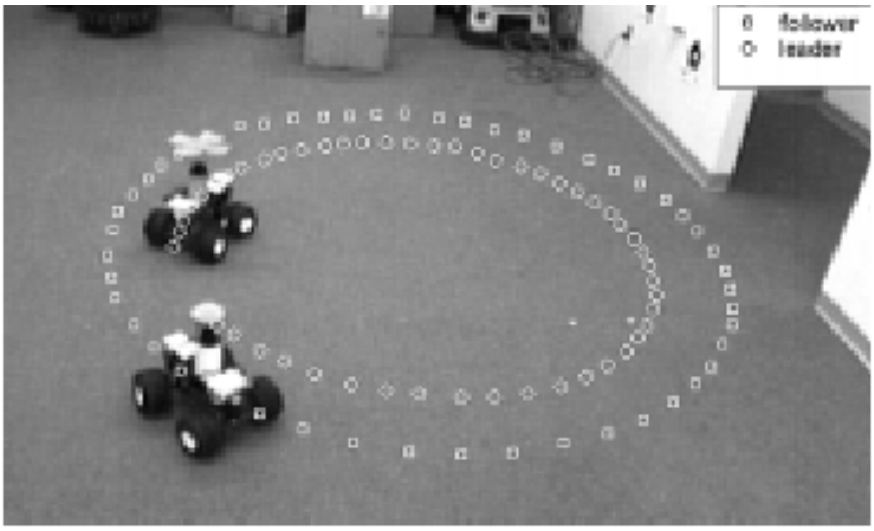

Figure 1.8. Leader Follower demonstration using an Omnicam for vision feedback as performed by the GRASP Lab

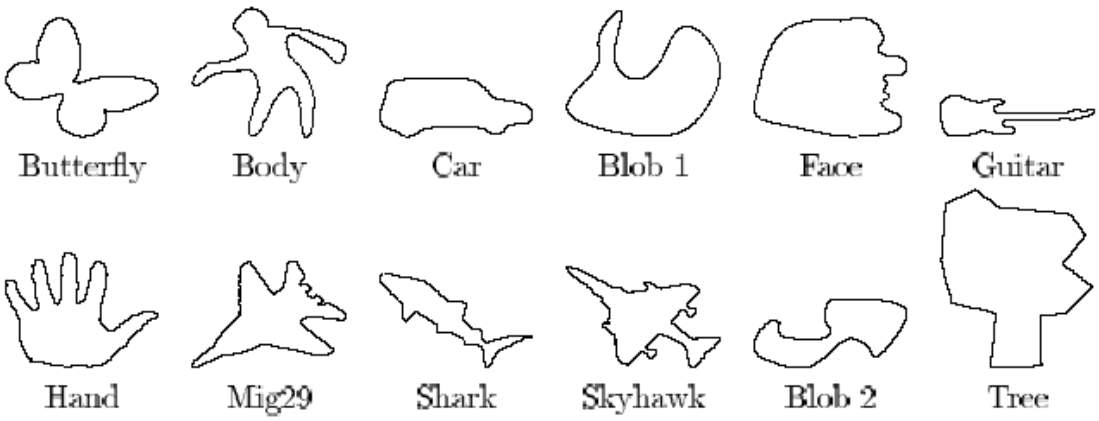

Figure 1.9. Examples of shape recognition for polynomial fitting

Choset and Kortenkamp [Cho03] describe the AERCam, which is an example of off board vision feedback. The camera is used with teleoperations to provide astronauts with visual information of the conditions and events outside the shuttle. The robot could

be used to provide distances between objects, which could be used eventually to provide information to robots for position control. Figure 1.10 shows a picture of the AERCam.

Figure 1.10. An astronaut with the AERCam from NASA Johnson Space Center

Dixon et al. [Dix01] presents another system with an off board camera system (figure 1.11). A fixed camera positioned above the robots in their workspace looks for LEDs or other specified physical characteristics such as colors or lines on the robots to determine their position and orientation. They further present a dynamical approach for control of the omnidirectional wheeled mobile robot using this position/orientation feedback. This off board method can be very beneficial since the processing of data can be done with a fast computer to control less sophisticated robots. Because an outside source must provide the global position/orientation to these robots, a study of communication is needed.

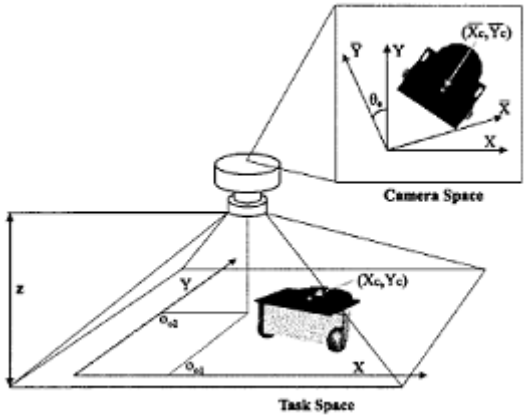

Figure 1.11. Off board camera positioned above a planar robot

1.3.2 Communication

Communication is essential in many of the previously mentioned multi-robot algorithms. Task allocation [Ost02], moving boxes [Don00], and playing soccer [Eme02] would not be possible without the proper communication. Two of the major decisions for communication in algorithms are knowledge sharing [Kur00] and commitment levels [Pir00]. Knowledge sharing refers to deciding what information is pertinent to disclose to the rest of the team. Commitment levels decide when and how the robots perform on an individual level as opposed to their work as team members.

Khoo and Horswill of Northwestern University have implemented a behavior-based system appropriately named HIVEMind, which allows each robot to consider the others as sensors [Kho02]. With a web of communicating robots, any robot in the network can access information from any robot. Because so many information packets must traverse through a web and creating sensory (information) overload, their teams of robots are limited to about ten members.

University of Pennsylvania and the Universidade Federal de Minas Gerais have collaborated to propose a method to allocate operations of each robot in the system under

a hybrid structure [Cha02]. This hybrid refers to limited amount of decision made by a central computer with some processing done on board the robot. Tasks and roles are assigned three ways, allocating, reallocating, and exchanging. A robot is allocated a new role after completing a task. Without completing a role a robot may begin a new role by reallocating. Robots may also trade or exchange role assignments. The difficult part of this algorithm is to decide when a robot should quit a particular task and assume another role. If efficiency can be measured, a robot should reallocate or exchange roles when it is performing poorly. Figure 1.12 shows a picture of the role assignments.

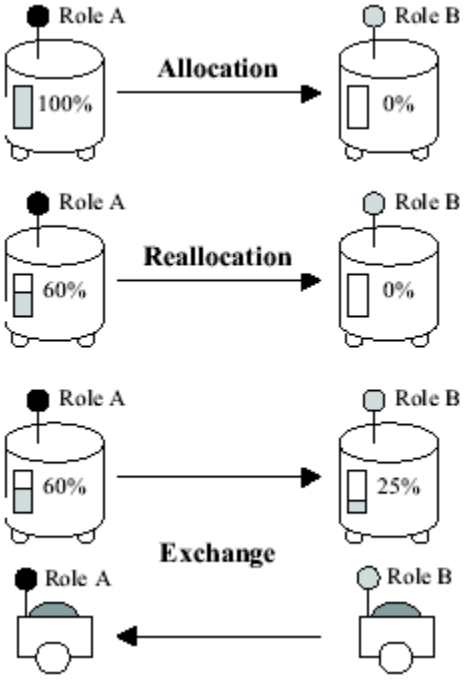

Figure 1.12. Dynamic role assignments (allocation, reallocation, and exchange of roles)

Brumitt and Stentz [Bur00] and Zlot et al. [Zlo02] with exploration emphasis have included methods for sharing maps. There are two popular methods for creating maps. The first is a pattern detection system that checks to see if common information is found between the robots' individual maps. Figure 1.13 represents such a map where the hall

and two doorways overlap between maps. The resultant of the combination is the map on the right with four rooms and a hallway.

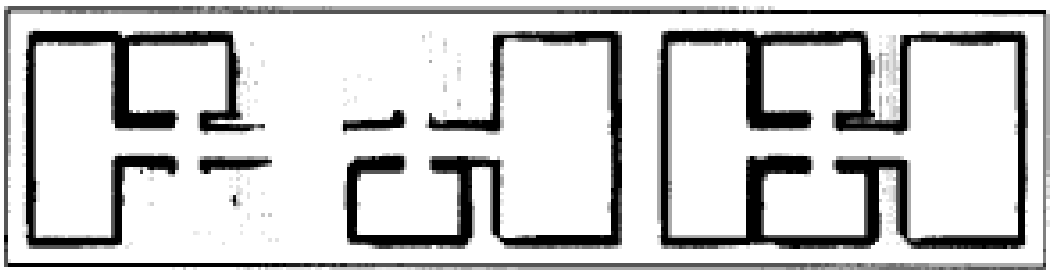

Figure 1.13. Partial maps created by individual robots using on board cameras are combined using a collective intelligence

The second method for combining maps that have been creating by several different robots is to know the position of the maps a priori in a global sense. Figure 1.14 shows a map where robots explored a territory and robot position feedback informed the map collector of their global position.

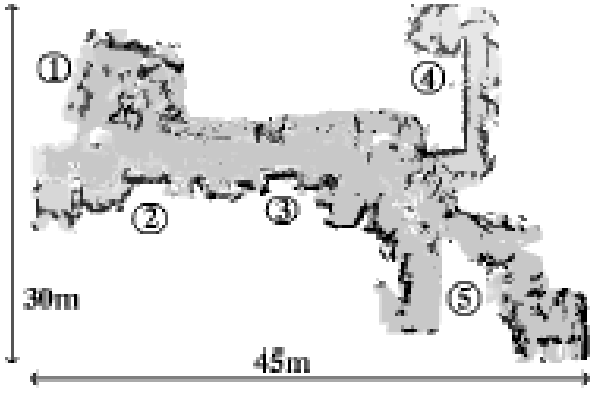

Figure 1.14. Multiple partial maps combined using a global coordinates

1.3.3 Parallel Steering

Most of the previously discussed algorithms have been tested only for omni-directional robots. However, there is a great interest in implementing these same algorithms on more robust vehicles such as the modern car. These nonholonomic robots,

as opposed to omni-directional robots, are restricted in their radius of curvature. An omni-directional robot can turn about its center, but a parallel steering mobile robot is constrained to turn with a minimum radius.

Laugier et al. ([Par96], [Par98], [Lau97]) discuss the constraints and necessary path planning for maneuvering a car. Figure 1.15 shows a few examples of progressing through an environment with walls. At each turn the vehicle must be able to account for a minimum radius of curvature. In addition to this capability to move around a structure, they have implemented a process to allow the car to parallel park. This takes advantage of the cars ability to move both forward and backwards.

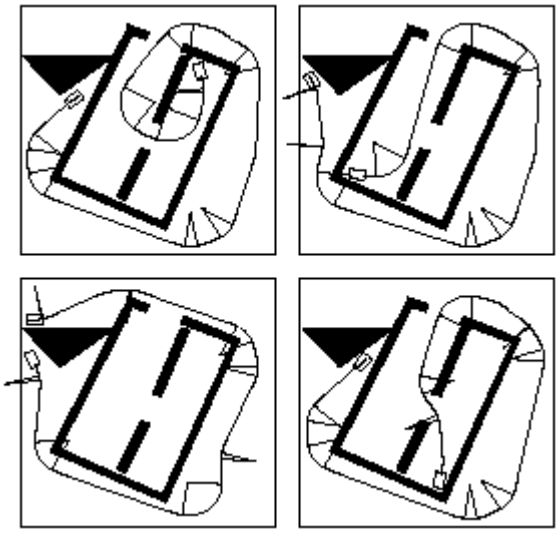

Figure 1.15. Four pictures of maneuvering around a structure with a minimum radius

Nagy and Kelly [Nag01] present methods for describing the motion of a car as a cubic curvature polynomial. This is displayed in figure 1.16. From a given position theoretically there are an infinite number of trajectory choices within a given range. Choosing the length of a given path and switching to a new path results in varying trajectories.

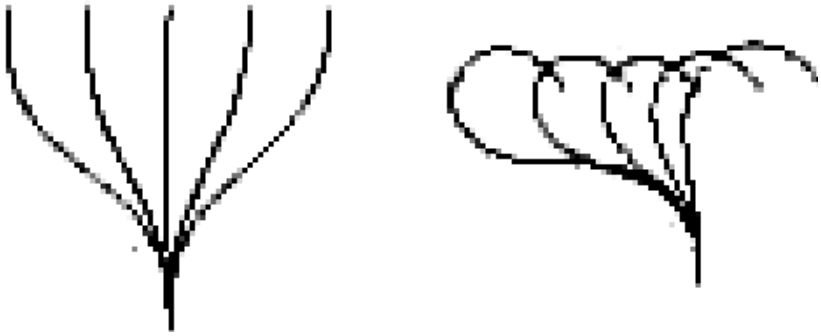

Figure 1.16. Examples of cubics with differing end postures

1.3.4 Path Planning

Path planning for a wheeled mobile robot describes how the vehicle will travel through an environment. There are two major path-planning algorithms; those that have knowledge about the workspace before traversing and those that do not.

Yahja et al. [Yah98] have implemented an algorithm called quadtrees that require knowledge about the environment prior to entering. While this quadtree method does not strictly require all of the information before entering the workspace, to conserve time it is most useful in that sense. Quadtrees as seen in figure 1.17 can describe the quadrant of a grid that gives a clear path toward the goal. Using the regular grid method on the left it can easily be seen how one should maneuver around the obstacle. The quadtree method on the right divides the entire workspace into fourths. The top half and the bottom right are all clear so they are not areas of particular interest. The bottom left quadrant where the obstacle is contained is then split into fourths and is again checked for obstacles. This is continually done until the robot has found a complete unblocked path in the chosen quads. This path is known as a quadtree. The greatest advantage of this method is that compared to the regular grid, the path planning requires little computation time. The best place to use quadtrees is in an open environment with a few large obstacles.

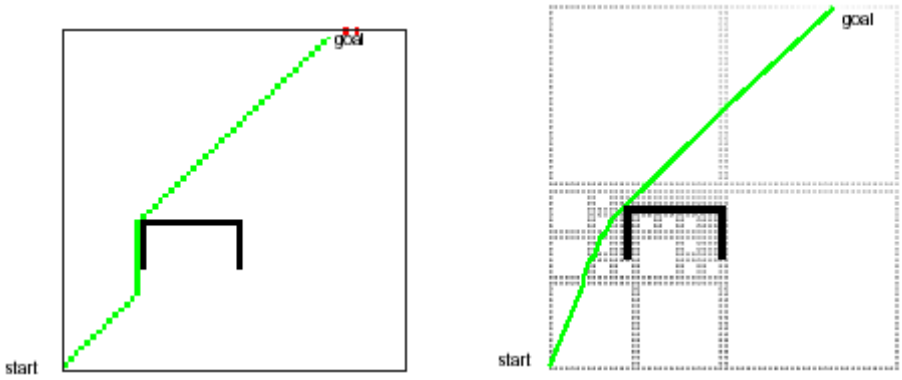

Figure 1.17. An example of path plans given by a grid (right) and quadtrees (left)

Burgard et al. [Bru00] have presented a mission planner for robots in a dynamically changing environment. When the settings of the robots' workspace are uncertain, a dynamically updating planner is needed. As a robot identifies new obstacles and new goals, the planner should revise its strategy and account for these new additions. Not only should a robot incorporate rectangular shaped objects such as walls and columns, but also incorporate a method to consider odd shaped objects and changes in elevation like hills and valleys.

Other important aspects of path planning include speed planning for the motion [Hwa02], optimal trajectory planning that includes smoothing splines [Ege01], and maneuvering while having obstacle detection from a laser range finder [Cha97]. Avoiding obstacles is another fundamental issue.

1.3.5 Docking/Latching

Docking is a broad term, which is often used to describe both the path planning to ensure proper alignment as well as the actual physical connecting between a robot and its destination. For this thesis, docking will (as it should) strictly refer to the placing together of any two matching parts. Latching further implies a physical linkage has been

set, which can be seen in figure 1.18 where the latch on the left grasps the handle on the right to attach itself.

Figure 1.18. An example of a latching mechanism to create a physical linkage

1.4 Proposed Methodology

Now that the state of the art architecture has been briefly discussed a synopsis of the methodology will be introduced. This approach will be used for the remainder of this study.

1.4.1 Knowledge of Coordinates

Position feedback will be determined using the PhaseSpace position measurement system. This is an off board camera system that tracks the 3D coordinates of each of the robots on a central computer. The positions are then fed to each robot as needed. This allows the robots to have a global coordinate system. Not having to incorporate a landmark detection system and image processing allows this project to focus on the path planning for docking and formation aspects of this thesis. The theoretical application for this system will be discussed more in section 2.1 and its equipment in section 3.1.

1.4.2 Communication

Communication among the robots will consist of collective intelligence with a limited hierarchy and extended sensory. This means that one member of the team, the camera system, will provide the rest of the collective with position information. The

robots will rely on the team goal and sensory information provided by each other to decide how to conduct their missions. The pertinent sensory data will be stored on a central computer. Each robot is free to access any kernel of information without the central computer forcing information. This approach is similar the HIVEMind previously discussed without the cluttered web of information held among the robots.

1.4.3 Path planning for parallel steering

One of the most important and therefore detailed areas of this research is path planning. By accessing the position data of each robot, end conditions are known to help create a 4^{th} order polynomial for the robot trajectories. Optimizations are performed to ensure that the length of the path is minimized and the turning radii conditions are satisfied.

1.4.4 Algorithm

The central focus of this thesis is to devise methods for determining a legitimate path between a robot and its goal. More importantly the desire to apply the design to multi-agents drives the implementation of the framework on the robotic team described in this work. Using the initial pose of the robots, a trajectory is planned to allow the robots to dock with one another.

Figure 1.19 gives a brief overview of the framework used for this path planning and tracking. A coordinate transformation from the camera frame is first performed to develop a planar coordinate system, in which the robots will travel. Knowing the initial pose of the robots, a trajectory between them is established and built as a function of time by introducing a desired speed along the path. The work of this thesis further applies the methods for determining a path by having the robots follow the selected trajectory through a series of experiments by tracking with classical control methods.

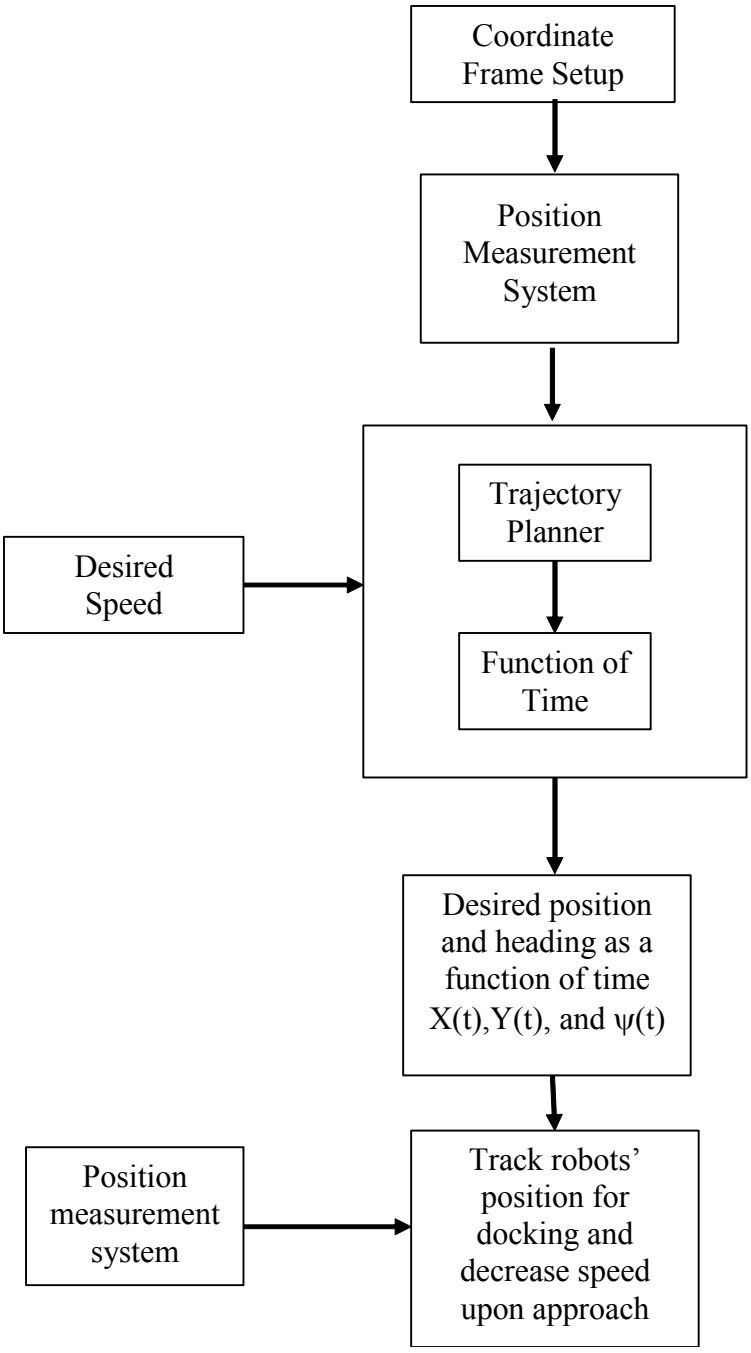

Figure 1.19. Algorithm for maneuvering of the robots en route for docking

CHAPTER 2
EQUATIONS AND ALGORITHMS

After introducing the state of the art algorithms and equipment currently being applied for multi-agent systems, it is pertinent to discuss the methodology for their implementation. An investigative discourse will be performed to adequately convey the steps taken for accurate path planning and docking for a multi-agent system consisting of two nonholonomic mobile robots. Included in this discussion are the position feedback, path planning, and steering and speed control algorithms.

2.1 Pose Feedback Algorithm

To acquire pose data for feedback, a set of cameras are used to determine the (x,y,z) position of an array of 10 light emitted diodes (LEDs). Five LEDs are fixed to the top of each robot in the pattern represented in figure 2.1, for defining body-affixed frames. The robots are tracked via the camera setup, which detects the position of the LEDs. The equipment for this setup is discussed in section 3.1.

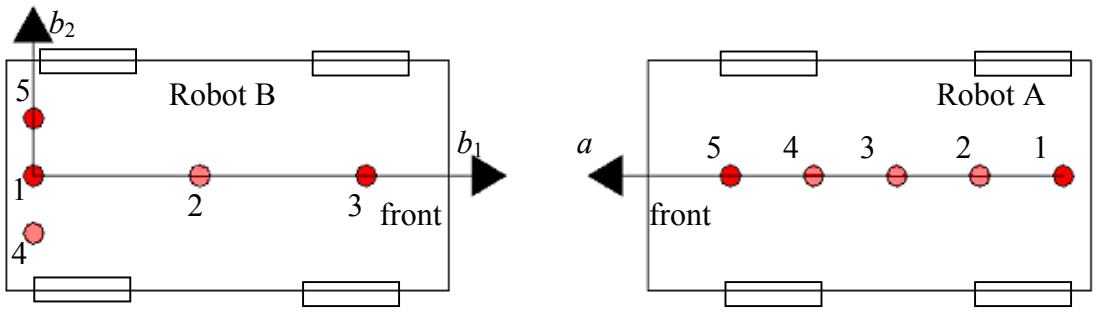

Figure 2.1. LEDs placed on top the robots and their directional vectors

To define frame axes, b_1 and a, LEDs, (#1 and #3) and (#1 and #5), are used to calculate the corresponding unit vector for robot B and A, respectively. Axis b_2 is determined using LEDs (#1 and #5). It is assume that the LEDs are mounted sufficiently accurate to generate an orthogonal set of axes, b_1 and b_2. It should be noted that while there are five LEDs on each robot only two are needed on Robot A (#1 and #5) and three on Robot B (#1, #3, and #5). Five LEDs were placed on each robot as a back up in case one of the initial pertinent LEDs was undetected; however, this sensor redundancy proved to be unnecessary in most cases.

The camera setup acts as a pseudo GPS where a global position of the LEDs is measured and recorded. In this thesis, the multi-agent robot actions are assumed to only involve motions on a planar surface, thus a transformation of the (*x,y,z*) coordinates to a planar representation is performed. At the initial positions, while the robots are stationary, the unit vectors for the $\hat{\underline{a}}$, $\hat{\underline{b}}_1$, and $\hat{\underline{b}}_2$ axes are calculated (equations 2.1-2.3). This is done by subtracting (*x,y,z*) position of two LEDs and dividing by the magnitude of their difference.

$$\hat{\underline{a}} = \frac{\left(x_{A_5} - x_{A_1}\right)\hat{\underline{g}}_1 + \left(y_{A_5} - y_{A_1}\right)\hat{\underline{g}}_2 + \left(z_{A_5} - z_{A_1}\right)\hat{\underline{g}}_3}{\left\|\left(x_{A_5} - x_{A_1}\right)\hat{\underline{g}}_1 + \left(y_{A_5} - y_{A_1}\right)\hat{\underline{g}}_2 + \left(z_{A_5} - z_{A_1}\right)\hat{\underline{g}}_3\right\|} \tag{2.1}$$

$$\hat{\underline{b}}_1 = \frac{\left(x_{B_3} - x_{B_1}\right)\hat{\underline{g}}_1 + \left(y_{B_3} - y_{B_1}\right)\hat{\underline{g}}_2 + \left(z_{B_3} - z_{B_1}\right)\hat{\underline{g}}_3}{\left\|\left(x_{B_3} - x_{B_1}\right)\hat{\underline{g}}_1 + \left(y_{B_3} - y_{B_1}\right)\hat{\underline{g}}_2 + \left(z_{B_3} - z_{B_1}\right)\hat{\underline{g}}_3\right\|} \tag{2.2}$$

$$\hat{\underline{b}}_2 = \frac{\left(x_{B_5} - x_{B_1}\right)\hat{\underline{g}}_1 + \left(y_{B_5} - y_{B_1}\right)\hat{\underline{g}}_2 + \left(z_{B_5} - z_{B_1}\right)\hat{\underline{g}}_3}{\left\|\left(x_{B_5} - x_{B_1}\right)\hat{\underline{g}}_1 + \left(y_{B_5} - y_{B_1}\right)\hat{\underline{g}}_2 + \left(z_{B_5} - z_{B_1}\right)\hat{\underline{g}}_3\right\|} \tag{2.3}$$

where x_{An}, y_{An}, z_{An} are the coordinates of the n^{th} LED on Robot A in the camera frame, n = 1,…,5

x_{Bn}, y_{Bn}, z_{Bn} are the coordinates of the n^{th} LED on Robot B in the camera frame, n = 1,…,5

$\hat{\underline{g}}_1, \hat{\underline{g}}_2, \hat{\underline{g}}_3$ are directional unit vectors in the camera frame {G} coordinates

$\hat{\underline{a}}$ defines the unit vector of the heading of robot A in the camera frame {G} coordinates

$\hat{\underline{b}}_1$ defines the unit vector of the heading of robot B in the camera frame {G} coordinates

$\hat{\underline{b}}_2$ defines a unit vector orthogonal to $\hat{\underline{b}}_1$ in the camera frame {G} coordinates

(e_1, e_2) is defined to be parallel to the axes (b_1 and b_2) at initial time, t_0.This frame's origin is calculated as a translation along the b_1 axis by L_{b1} from the origin of the (b_1, b_2) frame. That is, its origin is located at the front of Robot B at time, t_0. For path planning and tracking, the position and orientation of this frame is selected as the fixed frame of reference with an origin (0,0,0). Frame (e_1, e_2)'s origin and axes in {G} coordinates are defined respectively as

$$\hat{\underline{e}}_{1_{origin}} = \begin{bmatrix} x & y & z \end{bmatrix}^T_{A_1} + L_{b1}\hat{\underline{b}}_1 \qquad \text{at time, } t_0 \tag{2.4}$$

$$\hat{\underline{e}}_1 = \hat{\underline{b}}_1 \tag{2.5}$$

$$\hat{\underline{e}}_2 = \hat{\underline{b}}_2 \tag{2.6}$$

$$\hat{\underline{e}}_3 = \hat{\underline{b}}_3 \tag{2.7}$$

where L_{b1} defines the distance from the first LED on Robot B to the center front position of the vehicle along the unit vector $\hat{\underline{b}}_1$. The coordinate frame, E, is defined by the axes, (e_1, e_2, e_3).

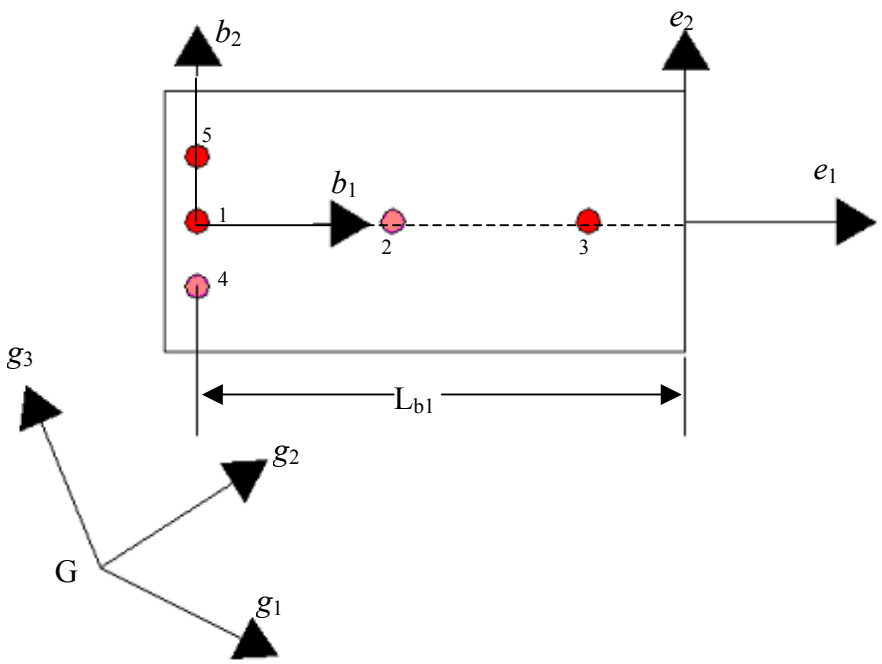

Figure 2.2 Frames E and B

2.1.1 Robot Position

Before Robots A and B can be tracked in the plane, proper definitions of the robots in the E frame's coordinate system must be included. The points of interest to be tracked are those associated with docking. I.e., because the docking occurs by gently touching the parallel fronts of the robots together, a virtual point at the center of the front edge of each robot will be tracked (figure 2.3).

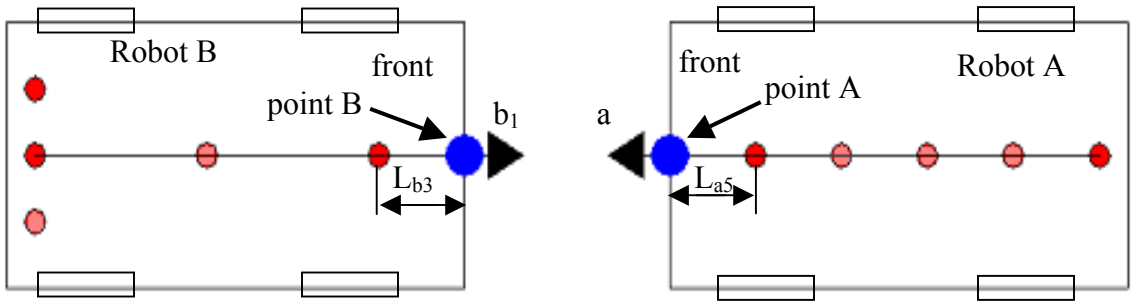

Figure 2.3 Point and heading to be tracked on each robot

The positions of the points of interest, A and B, in the camera frame are given by equations 2.8 and 2.9.

$$\underline{A} = \left({}^{G}x_{A_5}, {}^{G}y_{A_5}, {}^{G}z_{A_5}\right) + L_{a5}\hat{a} \tag{2.8}$$

$$\underline{B} = \left({}^{G}x_{B_3}, {}^{G}y_{B_3}, {}^{G}z_{B_3}\right) + L_{b3}\hat{b}_1 \tag{2.9}$$

where L_{a5} defines the distance from the fifth LED (closest to the front) on Robot A to the center of the front edge of the vehicle (Point A) along the axis, a, and L_{b3} defines the distance from the third LED (closest to the front) on Robot B to the center of the front edge of the vehicle (Point B) along the axis, b_1. The distance from the origin to each robot is calculated by creating vectors, $\underline{R}_A$ and $\underline{R}_B$, between the origin and the points of interest on Robot A and Robot B, respectively.

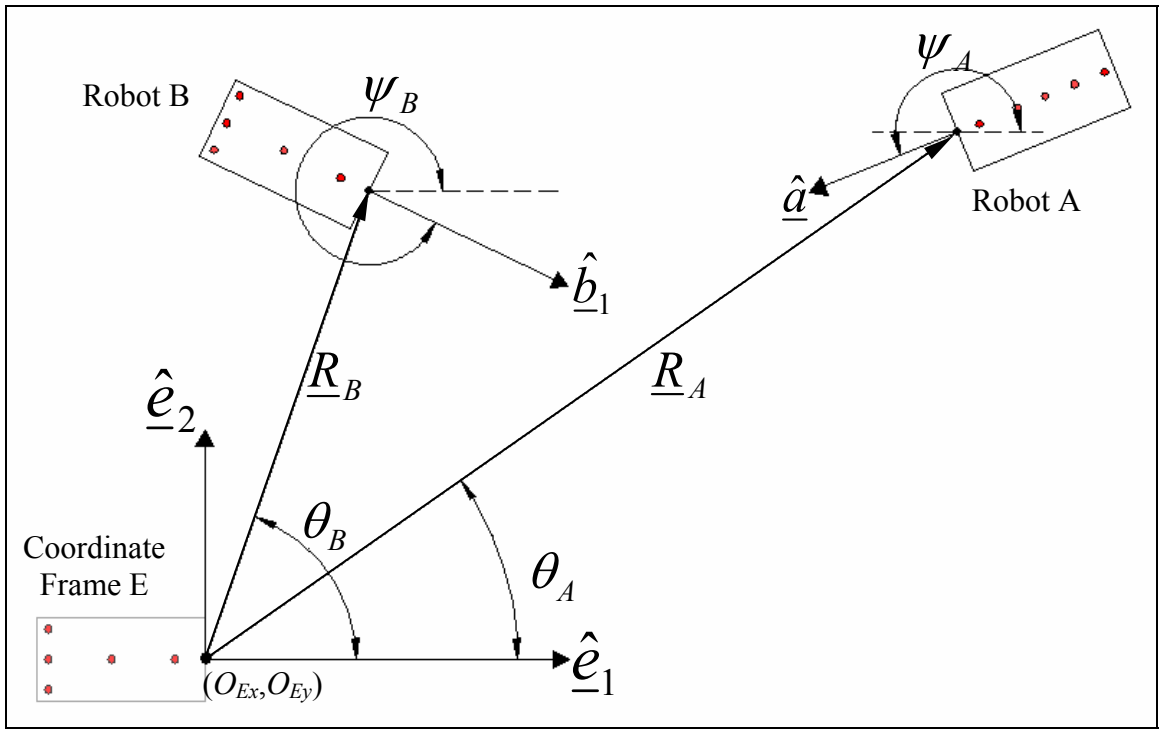

Figure 2.4. Robot A and Robot B in the E frame

Using the coordinates of A or B from the equations (2.8) and (2.9), $\underline{R}_A$ and $\underline{R}_B$ are defined in equations 2.10 and 2.11, and illustrated in figure 2.4.

$$\underline{R}_A = (A_x - O_{Ex})\hat{\underline{g}}_1 + (A_y - O_{Ey})\hat{\underline{g}}_2 + (A_z - O_{Ez})\hat{\underline{g}}_3 \tag{2.10}$$

$$\underline{R}_B = (B_x - O_{Ex})\hat{\underline{g}}_1 + (B_y - O_{Ey})\hat{\underline{g}}_2 + (B_z - O_{Ez})\hat{\underline{g}}_3 \tag{2.11}$$

At this point it is assumed that the motion lies in the (e_1, e_2) plane, which defines the plane of motion for a horizontal flat terrain. Therefore, the next step is to transform the datum to frame (e_1, e_2) of reference. First, the angles between the vectors, $\underline{R}_A$ and $\underline{R}_B$, and the unit vector, $\hat{\underline{e}}_1$, are needed. Using the dot product and multiplying the result by the sign of the vector, $\underline{R}_A$ or $\underline{R}_B$, dotted with $\hat{\underline{e}}_2$, yields

$$\theta_A = \cos^{-1}\left(\frac{(\underline{R}_A \bullet \hat{\underline{e}}_1)}{\|\underline{R}_A\| \bullet \|\hat{\underline{e}}_1\|}\right)\mathrm{sgn}(\underline{R}_A \bullet \hat{\underline{e}}_2) \tag{2.12}$$

$$\theta_B = \cos^{-1}\left(\frac{(\underline{R}_B \bullet \hat{\underline{e}}_1)}{\|\underline{R}_B\| \bullet \|\hat{\underline{e}}_1\|}\right)\mathrm{sgn}(R_B \bullet \hat{\underline{e}}_2) \tag{2.13}$$

where $((\ )\bullet(\ ))$ denotes the dot product of $(\ )$ and $(\ )$. It is preferred to track the robots in a rectangular coordinate system, thus the position of Point A defined in the planar rectangular coordinate frame of (e_1, e_2) is given by equation 2.14. Likewise the position of point B is given in equation 2.15.

$${}^E[x_A \quad y_A]^T = \|\underline{R}_A\|[\cos\theta_A \quad \sin\theta_A]^T \tag{2.14}$$

$${}^E[x_B \quad y_B]^T = \|\underline{R}_B\|[\cos\theta_B \quad \sin\theta_B]^T \tag{2.15}$$

2.1.2 Robot Heading

To perform the robot tracking along a selected path, it is vital to know the headings of each robot, which can be defined in terms of the unit vectors ($\hat{\underline{a}}$, $\hat{\underline{b}}$). Using the same

procedure as done above for θ_A and θ_B, equations 2.16 and 2.17 are expressions for the heading angles of robot A and B respectively, defined in (e_1, e_2) coordinate system.

$$\psi_A = \cos^{-1}\left(\frac{(\hat{\underline{a}} \bullet \hat{\underline{e}}_1)}{\|\hat{\underline{a}}\| \bullet \|\hat{\underline{e}}_1\|}\right)\mathrm{sgn}(\hat{\underline{a}} \bullet \hat{\underline{e}}_2) \tag{2.16}$$

$$\psi_B = \cos^{-1}\left(\frac{(\hat{\underline{b}}_1 \bullet \hat{\underline{e}}_1)}{\|\hat{\underline{b}}_1\| \bullet \|\hat{\underline{e}}_1\|}\right)\mathrm{sgn}(\hat{\underline{b}}_1 \bullet \hat{\underline{e}}_2) \tag{2.17}$$

Using the heading and position equations given by equations 2.14-2.17, a path for docking and control of each robot can commence. The remainder of this thesis will refer to position and heading defined in planar coordinates (e_1, e_2).

2.2 Path Planning Algorithm

The path planning of the two robots in the plane (e_1, e_2) begins with the knowledge of the initial pose (position and orientation) of each robot. Fitting a smooth trajectory between the two robots satisfies the two initial robot poses requires at least a third order polynomial. The robots also have parallel steering and therefore have a minimum turning radius of curvature. This minimum curvature applies a fifth constraint on the path. This condition suggests that a fourth order polynomial should be used. I.e.,

$$y(x) = \alpha_4 x^4 + \alpha_3 x^3 + \alpha_2 x^2 + \alpha_1 x + \alpha_0 \tag{2.18}$$

Because the initial position and orientation of Robot B at t_0 is used to define the inertial reference frame E's origin and orientation, Robot B's trajectory begins with values

$$y(0) = y_B(x_{B_0}) = 0 \tag{2.19}$$

$$y'(0) = y'_B(x_{B_0}) = 0 \tag{2.20}$$

where x_{B_0} = the initial x position of Robot B in the E frame coordinates, by definition of frame E, $x_{B_0} = 0$.

y_B = the y position of Robot B in the E frame coordinates

y'_B = the heading of Robot B in the E frame coordinates

This reduces equation 2.16 to following equation.

$$y(x) = \alpha_4 x^4 + \alpha_3 x^3 + \alpha_2 x^2 \tag{2.21}$$

The slope of the trajectory for Robot B at the position of docking with Robot A is defined by Robot A's heading (equation 2.22).

$$y'(x_{A_0}) = \tan(\psi_{A_0} - \pi) = 4\alpha_4 x_{A_0}^3 + 3\alpha_3 x_{A_0}^2 + 2\alpha_2 x_{A_0} \tag{2.22}$$

The constants α_4, α_3, and α_2 must now be determined. Using the initial position and orientation of Robot A, α_4 and α_3 can be written in terms of α_2.

$$\alpha_4 = \frac{y'(x_{A_0})x_{A_0} - 3y(x_{A_0}) + \alpha_2 x_{A_0}^2}{x_{A_0}^4} \tag{2.23}$$

$$\alpha_3 = \frac{4y(x_{A_0}) - y'(x_{A_0})x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^3} \tag{2.24}$$

where x_{A_0} = the initial x position of Robot A in the E frame coordinates

y_A = the y position of Robot A in the E frame coordinates

y'_A = the heading of Robot A in the E frame coordinates

The equation for the radius of curvature at each point, x, is given by equation 2.25.

$$\rho = \frac{\left(1 + y'(x)^2\right)^{\frac{3}{2}}}{|y''(x)|} \tag{2.25}$$

By inserting the derivatives of *y*, the turning radius of curvature becomes:

$$\rho = \frac{\left(1+\left(4\alpha_4 x^3 + 3\alpha_3 x^2 + 2\alpha_2 x\right)^2\right)^{\frac{3}{2}}}{\left|12\alpha_4 x^2 + 6\alpha_3 x + 2\alpha_2\right|} \tag{2.26}$$

$$\rho = \frac{\left(1+\left(4\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}^4}x^3 + 3\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^3}x^2 + 2\alpha_2 x\right)^2\right)^{\frac{3}{2}}}{\left|12\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}^4}x^2 + 6\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^3}x + 2\alpha_2\right|}$$

(2.26b)

Placing equation 2.27 in non-dimensionalized form where $x = \zeta x_{A_0}$ gives

$$\rho = \frac{\left(1+\left(4\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}}\zeta^3 + 3\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}}\zeta^2 + 2\alpha_2\zeta x_{A_0}\right)^2\right)^{\frac{3}{2}}}{\left|12\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}^2}\zeta^2 + 6\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^2}\zeta + 2\alpha_2\right|}$$

(2.27)

To keep the steering angle of the vehicles within it s turning radius capabilities, the remaining parameter α_2 is selected so as to maximize the minimum turning radius of curvature. A bisectional search can be used to determine α_2, searching over x and α_2 (equation 2.28).

$$\max_{\alpha_2}\left(\rho_{\min}\right) =$$

$$\max_{\alpha_2}\left(\min_{0\le x\le x_{A_0}}\left(\frac{\left(1+\left(4\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}^4}x^3 + 3\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^3}x^2 + 2\alpha_2 x\right)^2\right)^{\frac{3}{2}}}{\left|12\dfrac{y'\left(x_{A_0}\right)x_{A_0} - 3y\left(x_{A_0}\right) + \alpha_2 x_{A_0}^2}{x_{A_0}^4}x^2 + 6\dfrac{4y\left(x_{A_0}\right) - y'\left(x_{A_0}\right)x_{A_0} - 2\alpha_2 x_{A_0}^2}{x_{A_0}^3}x + 2\alpha_2\right|}\right)\right)$$

(2.28)

Once α_2 is determined then α_4 and α_3 can be calculated by substituting back into equations 2.23 and 2.24. Details of this search are presented later in this section. To control the position and orientation of each robot the desired position, (*x,y*), must be a function of time. By assuming a constant speed, maneuvering of the robots can be easily written in terms of time.

An additional constraint is introduced into the path planning that restricts the length of the curve between x_{B0} and x_{A0}. It is extremely important to add this limitation so as to not result in a trajectory causing vehicle maneuvering further away before moving toward its target where the length of the curve is much greater than the distance between the robots. This limitation can be a constant number or can be a function of the initial position. In this thesis, it was chosen to limit the length of the curve to 10 times the distance between the robots.

$$L_{x_B \to x_A} = \int_{x_B}^{x_A} \frac{1}{\sqrt{1+\left(\frac{dy}{dx}\right)^2}} dx < L_{\max} \tag{2.29}$$

where $L_{x_B \to x_A}$ = the length between the x positions of the robots

$L_{\max}$ = the maximum desired length of the path

Though it may seem as though the addition of this restriction over-constrains the problem, it actually bounds the range of α_2 values and therefore the variations in *y*. Further, it is theorized that by maximizing the minimum radius of curvature for a set of initial conditions, the length of the trajectory is minimized. Figure 2.5 provides an excellent example. The key trend to notice here is the radii of curvature versus the

lengths of the curves. As the path becomes more flat (large radii of curvature), the route also becomes much shorter. While most cases suggest that this hypothesis holds, it has not been rigorously proven and therefore is a condition that must be checked for each trajectory obtained in α_2.

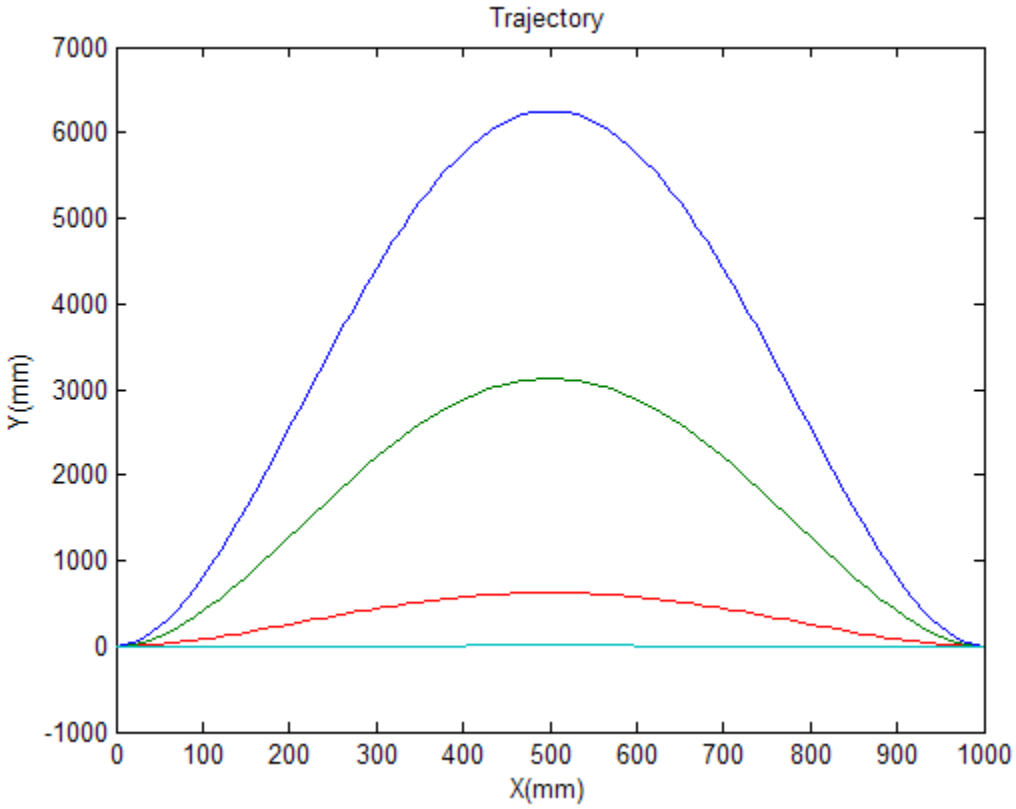

Figure 2.5. The minimum radius of curvature versus path length

Figure 2.6 summarizes the above analysis procedure used for solving the coefficients of the fourth order polynomial from the five constraints.

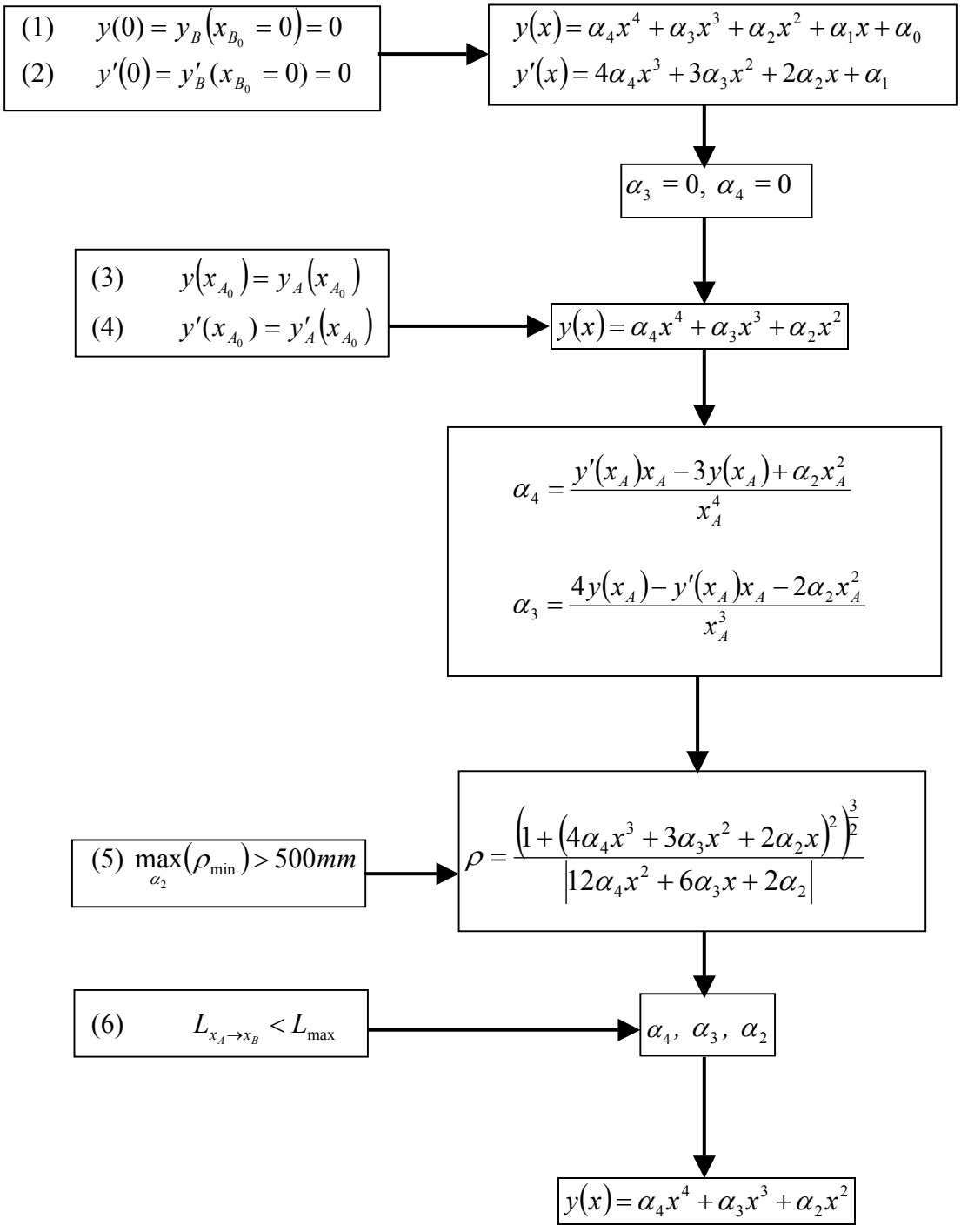

Figure 2.6. Algorithm for determining polynomial coefficients

The majority of the algorithm in figure 2.6 is rather straightforward, but the searching method for the value of α_2 needs further discussion. The bisectional search

algorithm is presented in figure 2.7. A bisectional search is implemented to determine the value of α_2 that maximizes $\rho_{\min}$ by driving the numerical derivative of $\rho_{\min}$ with respect to α_2, $\rho'_{\min}(\alpha_2)$,to zero, where ρ is as defined in equation 2.26b. Given initial guesses of c_1 (left bound) and c_2 (right bound), the $\rho'_{\min}(\alpha_2)$ is numerically evaluated and its sign checked at c_1 and c_2. If $\rho'_{\min}(c_1)$ is negative or $\rho'_{\min}(c_2)$ is positive, then the bisectional search has no solution for the given initial guess, bound on c. If $\rho'_{\min}(c_1)$ is positive and $\rho'_{\min}(c_2)$ is negative then a new c_1 is selected narrowing the bound on c. That is, c_1 moves half the distance to c_2. With the updated c_1 and c_2 values, the convergence of the search is checked by comparing c_2-c_1 to a selected error bound, ε. If the search is not terminated, then a new c_1 or c_2 is generated based on signs of $\rho'_{\min}(c_1)$ and $\rho'_{\min}(c_2)$, repeating the process until the convergence condition is met. If either c_1 or c_2 have crossed the maximum by changing signs of $\rho'_{\min}(c_1)$ or $\rho'_{\min}(c_2)$, then c_2 becomes c_1, and c_1 becomes the previous c_1. Otherwise, c_1 becomes half the distance to c_2. Then the criteria for length and minimum radius of curvature are checked. If both are satisfied, α_2 is set to c_1. The initial guesses converge to a maximum rather quickly (<<1 sec), but a study of the convergence rate should be performed before implementation in advanced levels of control of multi-robot systems. This algorithm does not incorporate values of α_2 outside its initial bounds, therefore the maximum must lie within the initial guesses, or a new initial guess must be selected. It is also important to note that these calculations are numerical solutions because a closed form solution to maximize $\rho_{\min}(\alpha_2)$ could not be symbolically obtained.

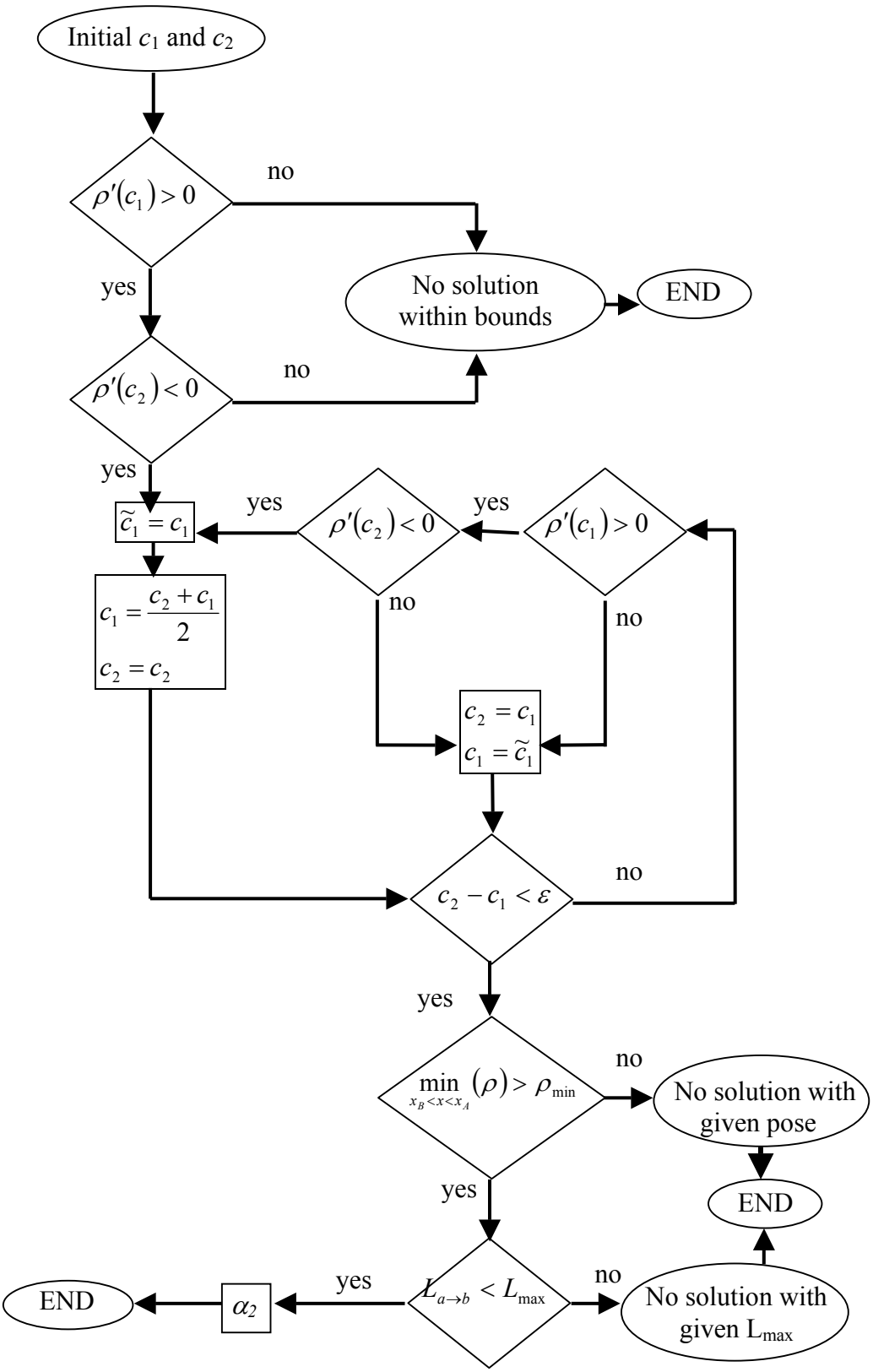

Figure 2.7. Searching Algorithm for finding α_2 and maximizing $\rho_{\min}$

Now that algorithm for finding α_2 has been presented, a brief example is given. By using end conditions x_A = 1500mm, $y(x_{A0})$ = -100mm, and $\psi(x_A)$ = -2.4 rads the plot of $\rho(\zeta, \alpha_2)$ results in the plot shown in figure 2.8. Where α_2 has initial guesses lying within (-100,100) ζ is the non-dimensionalized distance in x from Robot B to Robot A.

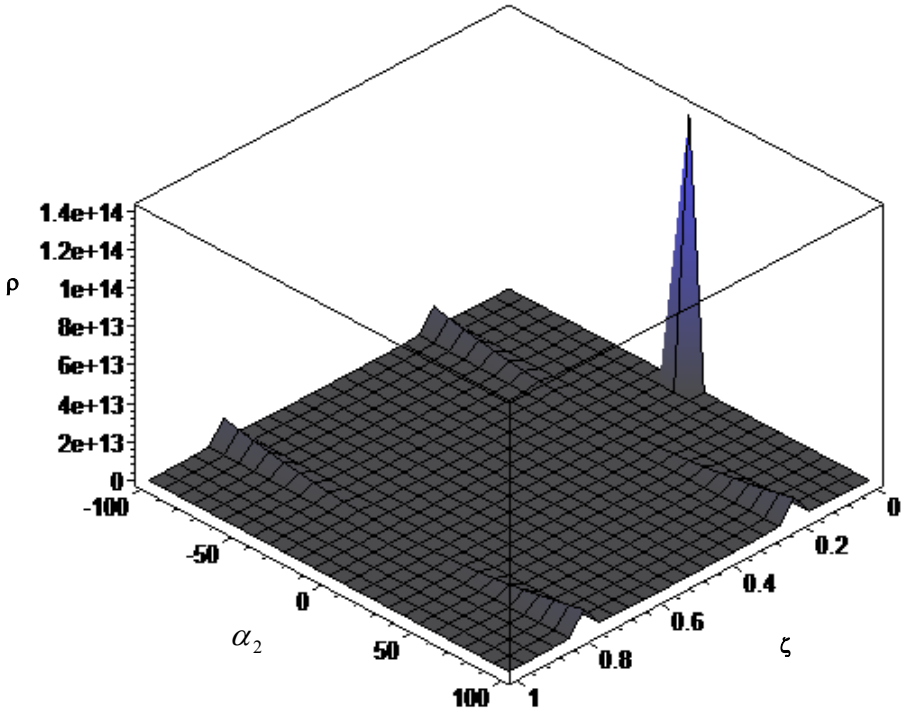

Figure 2.8. $\rho(\zeta, \alpha_2)$, initial guess -100<c<100, 0<ζ<1

The spike near (0,0) represents that the vehicle is basically traveling straight. The area of interest is not at the top of the plot but where the radius of curvature is very small. Figure 2.9 takes a slice of figure and removes the top portion and narrows the α_2 axis. It is important to note that this graph is a surface and not a solid; therefore, when the top is removed all of those areas with a radius of curvature greater than 600mm will not be displayed in figure 2.9.

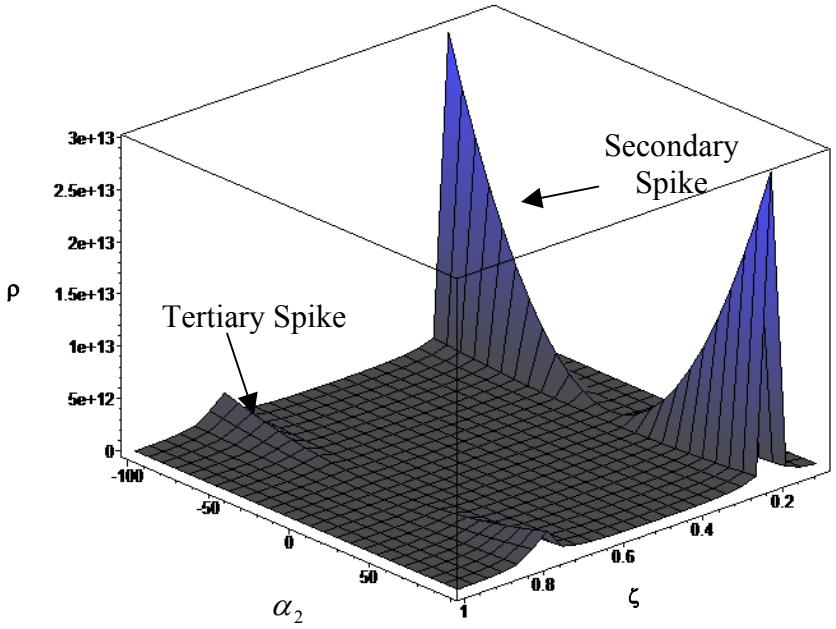

Figure 2.8b. ρ(ζ,c), initial guess -100< α_2 <100, 0.1<ζ<1 to exaggerate the secondary and tertiary peaks in magnitude which are rather symmetric about max(ρ_{min})

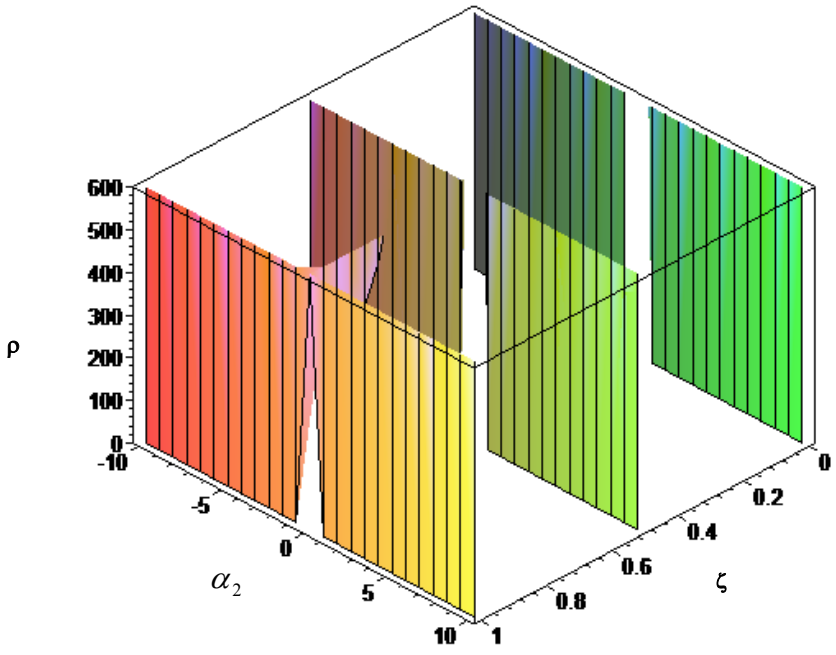

Figure 2.9. Sliced view of $\rho(\zeta, \alpha_2)$

By observing the critical areas pertaining to ρ_{min}, a view is taken from the α_2 - ρ plane, looking down ζ (figure 2.10).). The bottom edge of the shaded areas is ρ_{min} for each α_2 value.

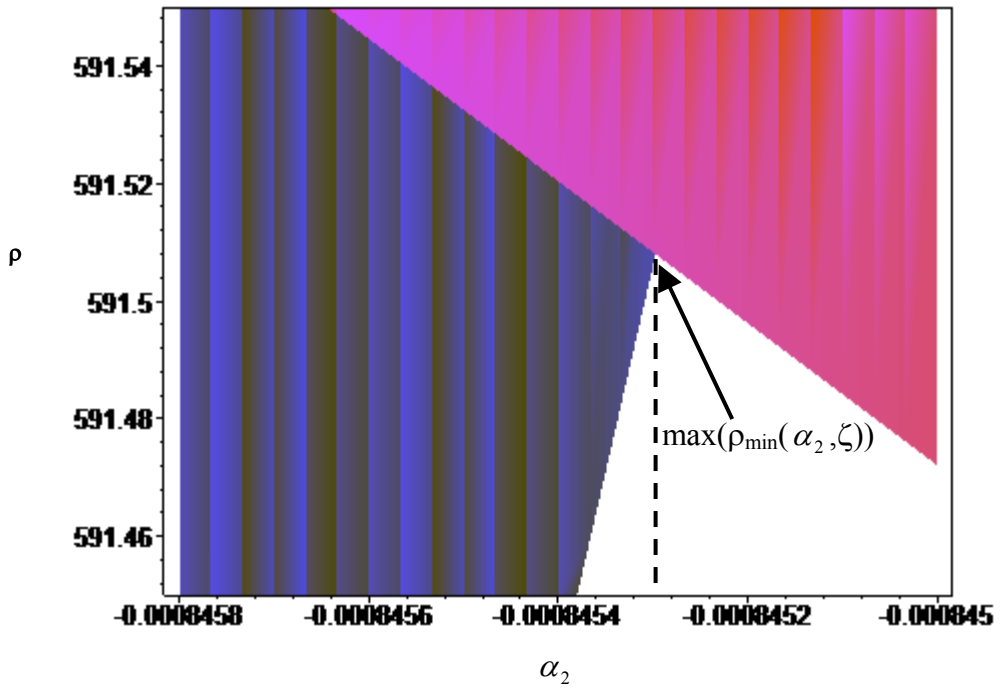

Figure 2.10. A view looking over all ζ in the α_2 - ρ to determine ρ_{min}

Using this projected view of the ρ values, the α_2 that yields the max(ρ_{min}) corresponds to the one that will exhibit the edge of the shaded areas. From figure 2.10, the α_2 value yielding max(ρ_{min}) for the example case is α_2 = -0.00084535. Comparing to the bisection method result an agreement of 99.994% was achieved. To complete the example, the plot of the trajectory comparing the different methods for achieving α_2 is given in figure 2.11. With the high percentage of agreement between the two methods for searching for α_2, their plots lie directly on top of one another. Further, it does not appear by looking at the path that the minimum radius of curvature could have been made

any larger thus giving credence again to the implemented bisection method. Figure 2.12 shows the plot of radius of curvature along ζ.

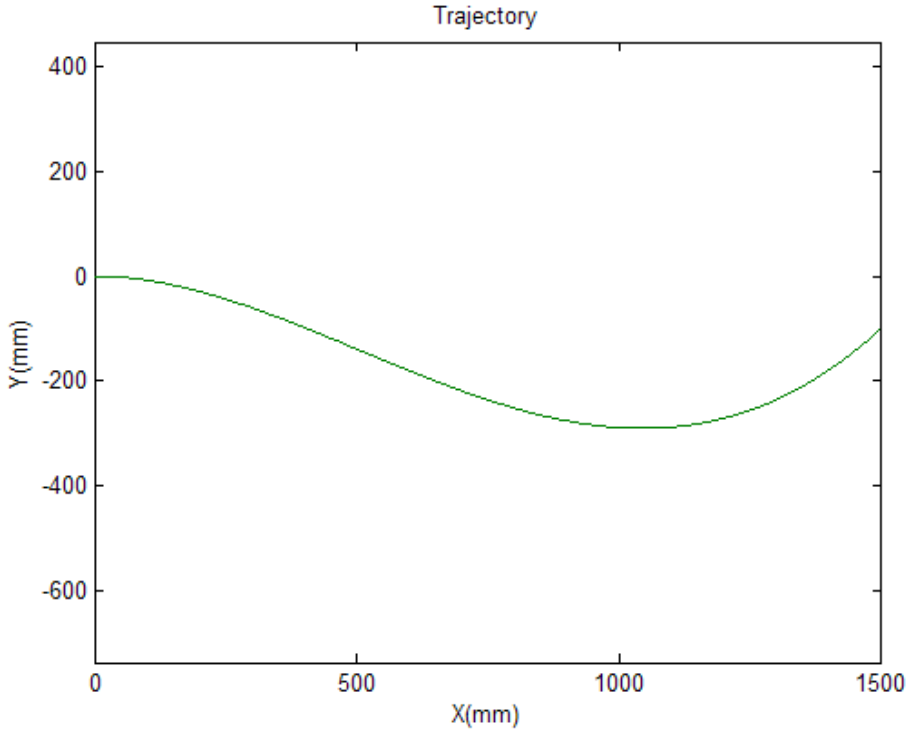

Figure 2.11. Trajectory, $Y(x)$, for the α_2 value selected by the bisectional method versus that determined through the exhaustive search. MatLab , which was used to plot the curves, could not distinguish between the trajectories.

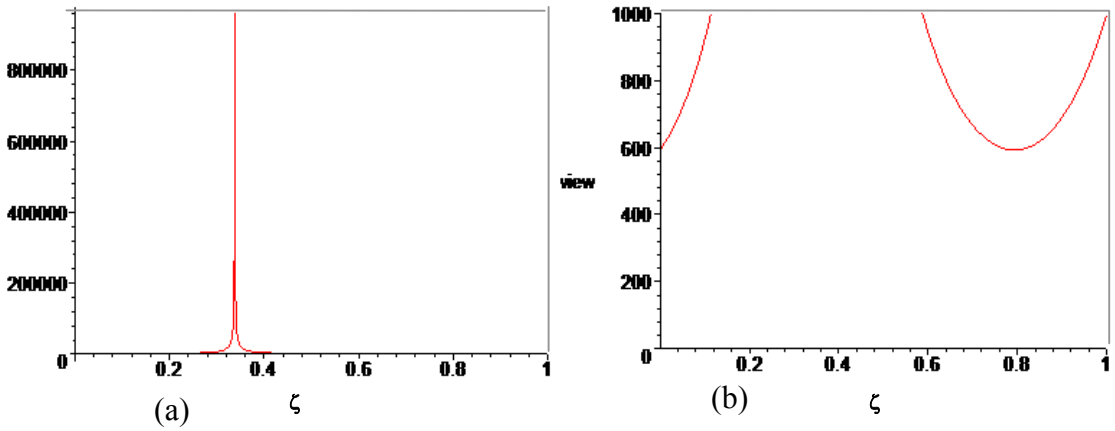

Figure 2.12. Radius of curvature $\rho(\alpha_2, \zeta)$ for α_2 = -0.00084530 (Exhaustive search method) (a) includes the entire data set and (b) focuses on the smaller radii of curvature by cropping the top

Table 2.1 shows a summary of various path-planning cases, illustrating smooth curves with minimal L_{max} as well as cases in which a curve could not be found to meet the criteria.

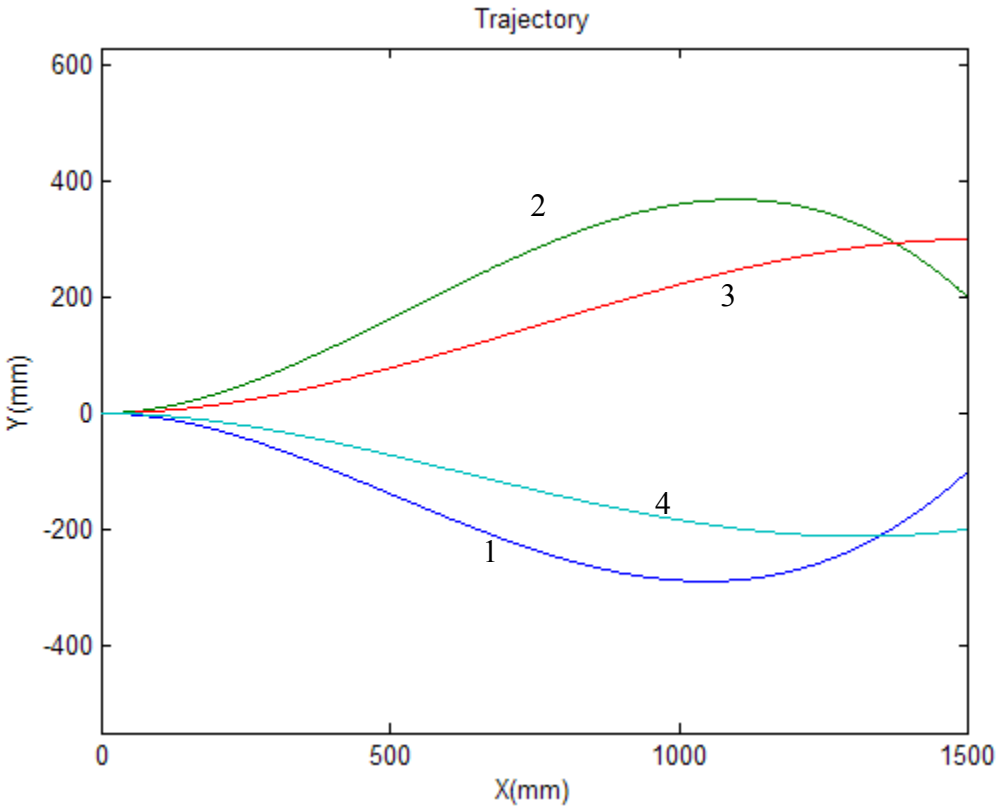

Figure 2.13. Various case trajectories (supplemented by table 2.1). Case 5, which is unacceptable, is not shown.

Table 2.1. Summary of case trajectories included α_2 values and criteria conditions

	α_2 (Bisection search)	α_2 (Exhaustive search)	% Agree	L	ρ_{min}	Criteria check	Number of iterations
1	-0.00084536	-0.00084530	99.993	1598	591	Y	31
2	0.00096149	0.00095617	99.444	1616	520	Y	39
3	0.00040001	0.00040000	99.998	1535	1250	Y	43
4	-0.00040305	-0.00040304	99.998	1521	1241	Y	30
5	0.00070889	0.00078976	89.760	1788	426	N	39

To achieve the time it takes to reach a point along the curve, t(x), the following definition can be used. Let t(x) be a function that relates the distance along the e_1 axis, x

such that the length, L, along the trajectory and the desired speed of the vehicle are also a function of *x*.

$$\delta L = \sqrt{(\delta x)^2 + (\delta y)^2} \tag{2.30}$$

$$speed_{desired} = \frac{\delta L}{\delta t} \tag{2.31}$$

$$t(x) = \int dt = \int \frac{dL}{dspeed_{desired}} \tag{2.32}$$

$$t(x) = \int_{x_B}^{x} \frac{L_{x_B \to x}}{speed_{desired}(x)} dx \tag{2.33}$$

A similar approach can be used in *y*, however, *x* is monotonically increasing by the *y*(*x*) that was generated in the previous section. Thus, it ensures that the time variable will also be monotonic. For more complex motions such as looping the approach will need to be modified to accommodate both *x* and *y*. This has been left for future work. By assuming a constant speed, maneuvering of the robots then becomes simply a function of the instantaneous path length measure from x_B (t_0) (equation 2.33) divided by the desired speed of the vehicles.

The path-planning algorithm of the previous section is to be the basis for a trajectory planner for an overall control system of multi robot systems. However, before it can be implemented effectively, the trajectory information must be converted to parametric form of ($X_A(t)$,$Y_A(t)$) ($X_B(t)$,$Y_B(t)$). The complexity of the overall algorithms are to kept at a minimal so as to realize a realtime implementation with inclusion of advanced capabilities such as dynamic obstacle avoidance, increased number of robots with all robots moving in formation and/or performing cooperative tasks, and operating in uncertain environments with limited sensing and actuation. Thus, the underlying

problem addressed in this thesis is the implementation of the previous section's path planning using a simple introduction of time variable. This yields an expression for time as a function of x that can be used in the trajectory algorithm consisting of speed and steering controllers.

2.3 Speed Update Algorithm

Proper speed is critical to control the rate of the error of the position and heading of each robot. To accurately control the vehicles, the speed and furthermore the motion of each vehicle needs to be smooth. At the same time, to properly follow the desired trajectory, the delay to reach the commanded position must be minimal. A fast reaction (quick rise time) often causes an overshoot, which is undesirable. A limit is placed on the speed so as to prevent the robot from going into reverse should it get ahead of its commanded position in terms of X(t). In severe cases, the drive motor is turned off while waiting for the commanded position to catch up in X. This setting can often cause oscillations, which too are undesirable. Stopping and starting can create an even larger lag because of the time it takes to regain speed. It can be noticed that to consider a constant speed throughout the trajectory, there should be an infinite acceleration at the start of the path since each robot begins at rest. This same principle is seen throughout the course of each experiment when a robot pauses because it leads it commanded position and cannot accelerate quickly enough once it again lags the commanded, X(t).

A proportional derivative (PD) controller attempts to keep the robot operating at its nominal speed. Hence, it is used to increase the speed of the vehicle when the robot falls behind the value of the commanded, X, at an instant in time as well as reduce the approach speed when the vehicle nears its target position. Figure 2.14 shows a closed loop speed control system of the robots.

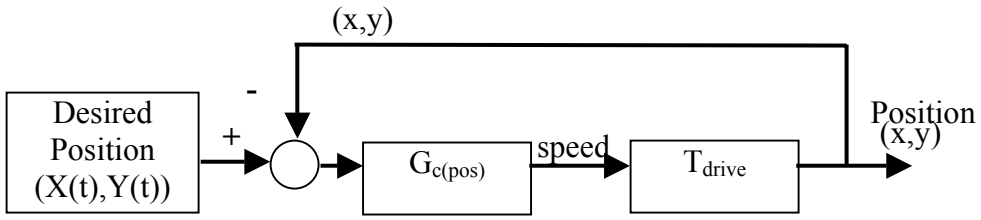

Figure 2.14. Closed loop speed controller with position feedback

The $G_{c(pos)}$ denotes the PD compensator, T_{drive} represents the transfer function of the rear motor, and the camera data is fed back into actual position and heading of the robots.

An initial implementation of the speed controller demonstrated the ability to determine the error between the commanded position along the trajectory and its actual distance along the predetermined path. However, using just the position error function exhibited difficulties in bringing the robot back on track both in X and t when it deviated from the path. To overcome this tracking problem, the following error and command signals were used. The error used for the speed calculations is the magnitude of the robot's distance between its the commanded and actual positions. A sign function is incorporated to specify whether the robot is ahead of its target location (equation 2.34). The speed command sent to the drive motor is given in equation 2.35 where the control constants are included.

$$Error_{speed} = \sqrt{(X-x)^2 + (Y-y)^2} \cdot \mathrm{sgn}(x-X) \tag{2.34}$$

$$speed = k_{pspeed} Error_{speed} + k_{dspeed} \frac{\Delta Error_{speed}}{dt} \tag{2.35}$$

where k_{pspeed} = proportional control constant for the error from current desired position

k_{dspeed} = derivative control constant for the change in error of position

In the implementation of the path planner for two moving robots (A and B) that are required to dock together, the same trajectory is used for both robots. Robot A starts from x_{A0} approaching x_{B0}, and Robot B starts at x_{B0} approaching x_{A0}. An alternate control mode of the speed controller is implemented when both robots are approaching each other from opposite ends of the polynomial path. The controller action is a function of the distance between the robots. If this controller mode is not applied then the robots will move along the trajectory until they collided with each other. The robots can now safely advance along the path and consider the relative position of the other robot in its speed control. In this controller the error is calculated by simply replacing the commanded position with the position of the other robot as seen by comparing equations 2.34 and 2.36.

$$Error_{speed} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \cdot \mathrm{sgn}(x_A - x_B) \tag{2.36}$$

Because the speed of the sensory data is limited, a safety measure is taken to ensure the robots do not collide. The robots are commanded to stop when they are within a prescribed distance from their target. The drive motor does not move until a minimum voltage is applied. The limitations on the sensors and minimum speed setting for the drive motor make this implementation necessary. The sensory discussion is continued throughout this work.

2.4 Steering Update Algorithm

With the speed properly controlled, it is pertinent to include an algorithm such as to minimize the position error and the heading error at all times. The steering needs to react quickly enough as to not cause the robot to miss its target, but not so fast as to hinder its use in future (real world) applications. Figure 2.15 shows the use of a closed loop

controller to specifically minimize the position error (equation 2.37) caused by overshoots and steady state error for a robot operating under speed control alone.

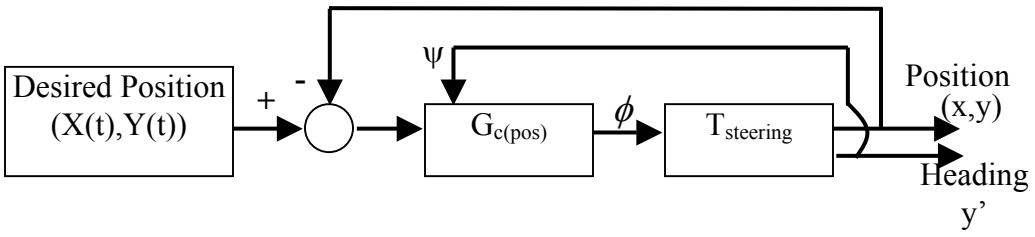

Figure 2.15. Closed loop controller for the position of the vehicles

Having position feedback and therefore knowledge of the error associated with the robots position, a proportional derivative compensator defined by equation 2.38 is implemented.

$$Error_{(x,y)} = \left(\tan^{-1}\left(\frac{Y-y}{X-x} \right) - \psi \right) \tag{2.37}$$

$$\phi = \phi_0 + Error_{(x,y)} k_{p(x,y)} \frac{\Delta Error_{(x,y)}}{dt} k_{d(x,y)} \tag{2.38}$$

where (X,Y) = the desired position of the robot, which changes with time

(x,y) = the vehicles current position in the E frame

ϕ_o = center angle of the steering

ϕ = steering angle measured via PWM

dt = the change in time between measurements (not considered constant)

$k_{p(x,y)}$ = control constant for the heading and vehicle position

$k_{d(x,y)}$ = control constant for the heading and vehicle position

Figure 2.16 illustrates the controller action. E.g., given the error shown in the figure, the controller would cause the steering angle to be changed to thus bring the robot back to the nominal path.

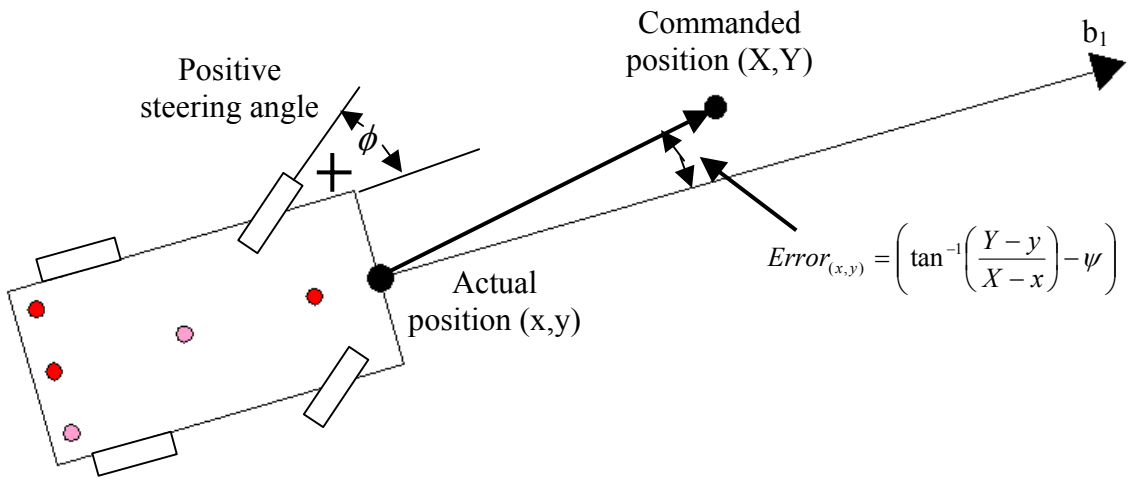

Figure 2.16 Example for the steering controller of the position of the vehicle

While the use of the controller for position indirectly controls the heading of the robot it does not guarantee proper alignment at its final position, which is extremely important. Figure 2.17 shows a closed loop controller that helps eliminate the heading error (equation 2.39). Feedback, which includes the information about the robot's heading, is fed to the compensator. This allows for proper update of the steering mechanism.

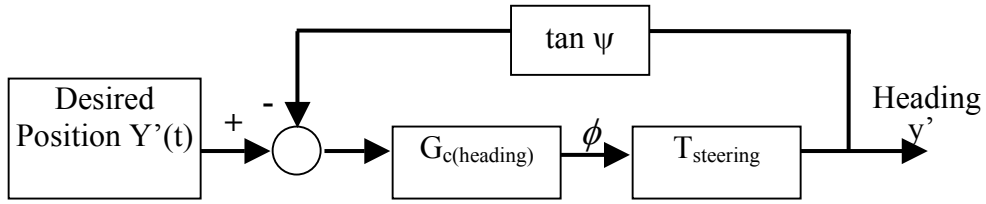

Figure 2.17. Closed loop steering controller for the heading of the vehicles

Equation 2.40 is used to control the error from the robot's heading. This too is a PD controller.

$$Error_{\psi} = \left(Y' - \tan\psi\right) \tag{2.39}$$

$$\phi = \phi_0 + Error_{\psi} k_{p\psi} + \frac{\Delta Error_{\psi}}{dt} k_{d\psi} \tag{2.40}$$

where:

Y′ = the current desired slope

$k_{p\psi}$, $k_{d\psi}$ = PD control constant for the heading and slope of polynomial

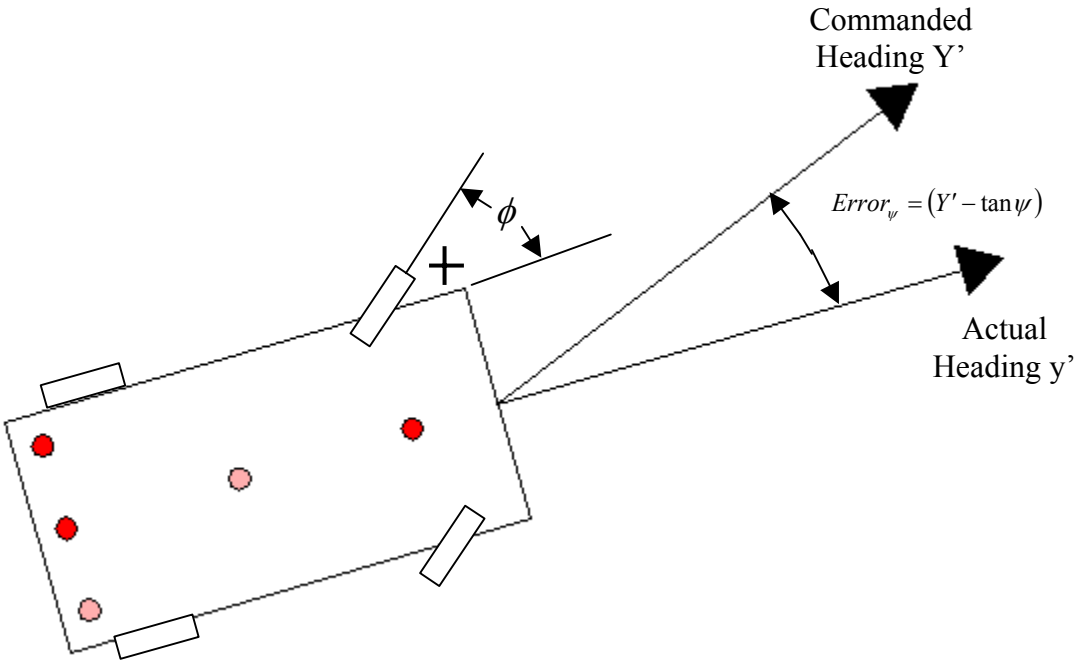

Figure 2.18. Example for the steering controller of the heading of the vehicle

Implementation and results delineating the performance of each of these controllers and the overall controller are given in chapter 5.

Because neither of the controllers described above have a guarantee for both heading and position a combination of the two is necessary. Figure 2.19 shows how this algorithm is implemented. The basic of this formula adds the compensation of the previous algorithms and causes improved reactions in both position and heading. Equation 2.35 is a combination of 2.38 and 2.40.

$$\phi = \phi_0 + Error_{(x,y)} k_{p(x,y)} \frac{\Delta Error_{(x,y)}}{dt} k_{d(x,y)} + Error_{\psi} k_{p\psi} + \frac{\Delta Error_{\psi}}{dt} k_{d\psi} \qquad (2.41)$$

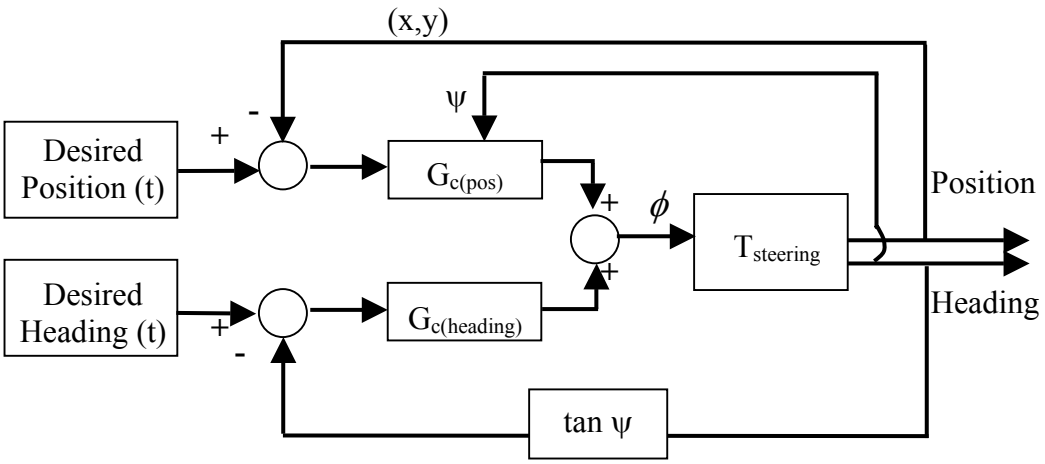

Figure 2.19. Controller for a combination of the Head and Position of the vehicles

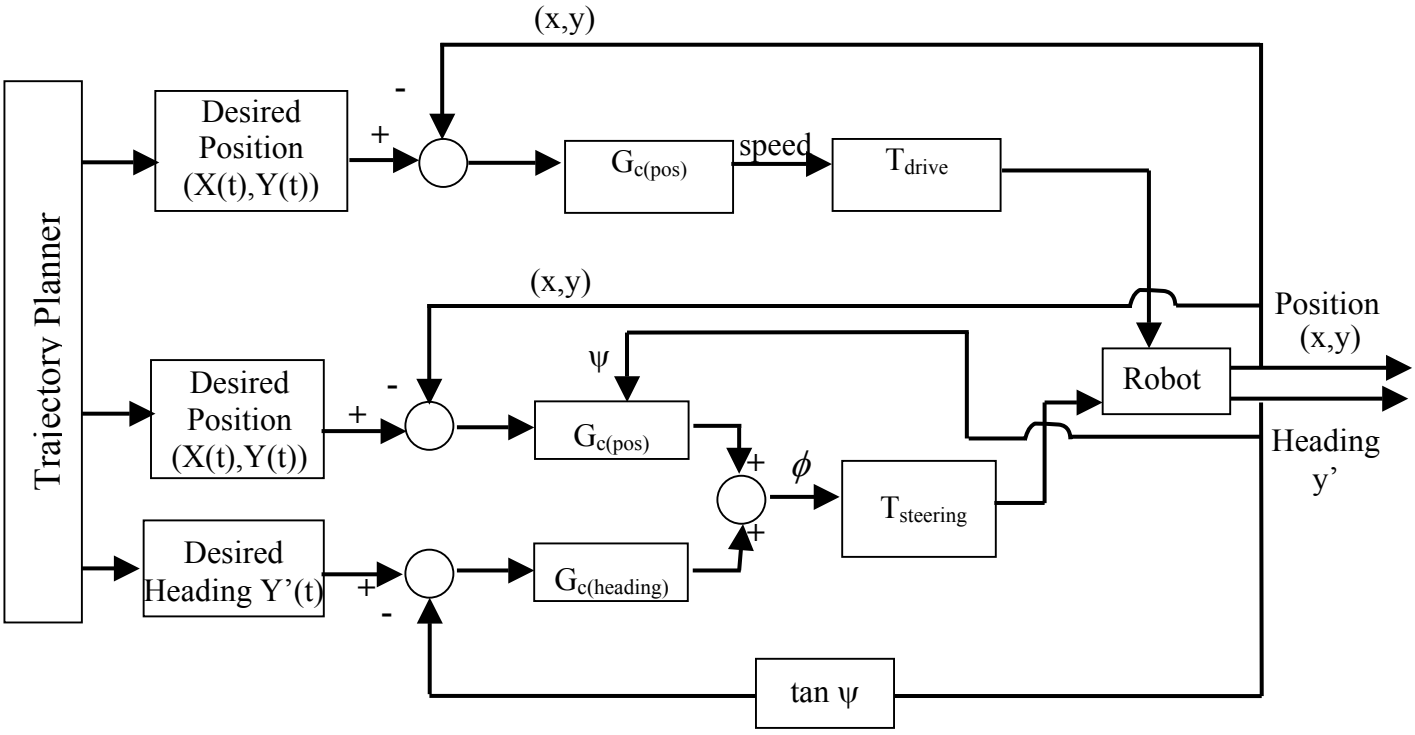

Figure 2.20. Overall controller for a combination speed and steering of the vehicles

CHAPTER 3
EQUIPMENT

In this chapter, a brief description of the equipment and testbed is presented. The experiments are performed in a testbed of cameras for position feedback and two mobile robots. The components of robots and the position measurement system and their applications will be discussed.

3.1 Phasespace Camera System

The following description of the camera system will be in terms of the mechanical/ electrical equipment and the arrangement of the cameras and the implications of the setup. Examples will also be given as to what the data looks like and its terminology in this paper.

3.1.1 Equipment

The camera system (table 3.1) consists of four CCD cameras (figure 3.1) that track the 3D position of 10 LEDs using triangularization. A minimum of two cameras is needed to track an LED. Four cameras are used here to increase the reliance that the LEDs will indeed be tracked. If a camera's view is blocked from the LED, it obviously does not detect it. After calibrating the cameras, the position and orientation of the cameras with respect to the camera labeled as "B" is known. The CCD cameras measure the distance to an LED with a 60^{o} field of view. If another camera sees the same LED, its measurement is also taken. Then knowing the relative position between the two cameras, the (x,y,z) position of the LED is known. The accuracy of the measurements decreases with about 1mm of accuracy per meter as the distance to the furthest camera increases

Table 3.1. Serial numbers of the components of the Phasespace position measurement system

Category	Description	P/N	S/N
Computer	AMD Athelon XP 2000+ 512MB memory SlackwareLinux 8.1		
Hub	Hub1-PSMC1	910-00001	1000003
Camera	Camera A	S/N: D3000008338CSA01	
	Camera B	S/N: E80000083386CB01	
	Camera C	S/N: 45000008339A8C01	
	Camera D	S/N: 2D000008338F4F01	
Box	Box 20 LBRD5	910-00013	1000006
	Box 21 BRD5	910-00013	1000007
	Box 22 BRD5	910-00013	1000008
Port Expander	LPX1	910-00015	1000002
Calibration	Calib-F3 PSMC1	910-00011	1000002
	Calib-C PSMC1	910-00010	1000002

Figure 3.1. The four identical cameras and a close up picture of one camera from Phasespace, Inc.

The LEDs operate at different frequencies, which allows them to be tracked independently. However, if the LEDs are too close to one another, only one will be detected. Other causes for not detecting an LED are when the LED is blocked from the cameras view or when the LEDs are not observed by more than one camera.

3.1.2 Setup

The cameras must be setup to capture the maximum coverage area while maintaining high accuracy. With the current setup, the camera furthest away from the workspace is about 3.6m. This results in better than 5mm of accuracy. Figure 3.2 shows the setup of the cameras around the workspace. The workspace measures nearly 1m by 1m. As one can see, the distance away from the workspace is large compared to the size of workspace. Figure 3.3 shows the area of coverage (field of view) from the cameras perspective.

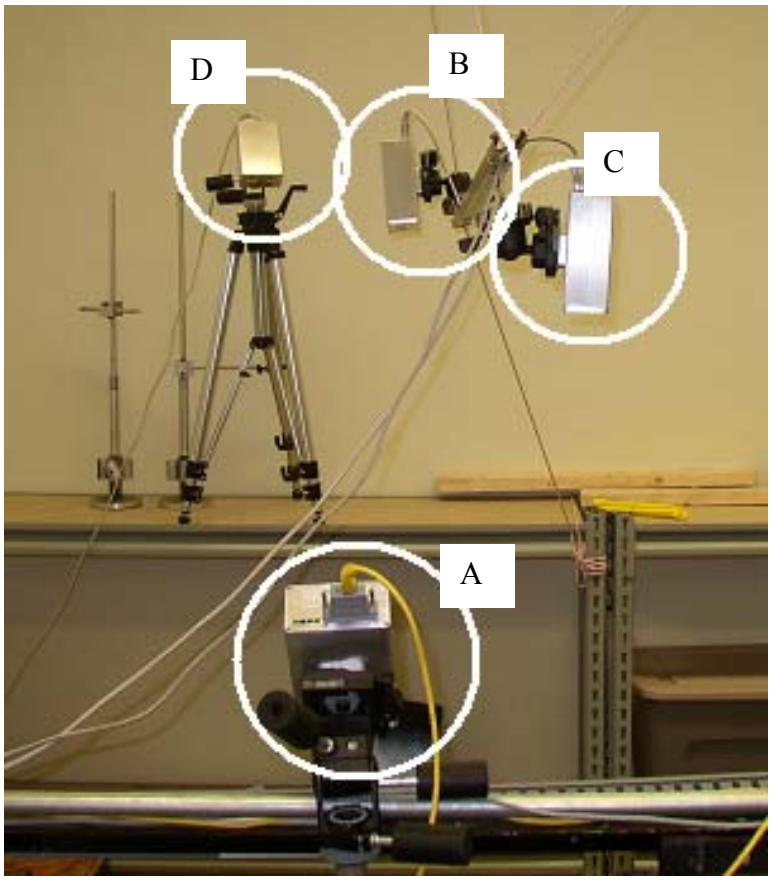

Figure 3.2. Camera setup

As mentioned in chapter 2 the arrangement of the LEDs is important to the position information. Several tests were run to determine how far apart the LEDs should be

placed. Figure 3.4 shows the final choice for their placement. The farther the LEDs are away from one another results in better accuracy of the position of the vehicle. In addition, the point being tracked is the center point along the front edge of the robot thus it was important to place an LED as close as possible to that location.

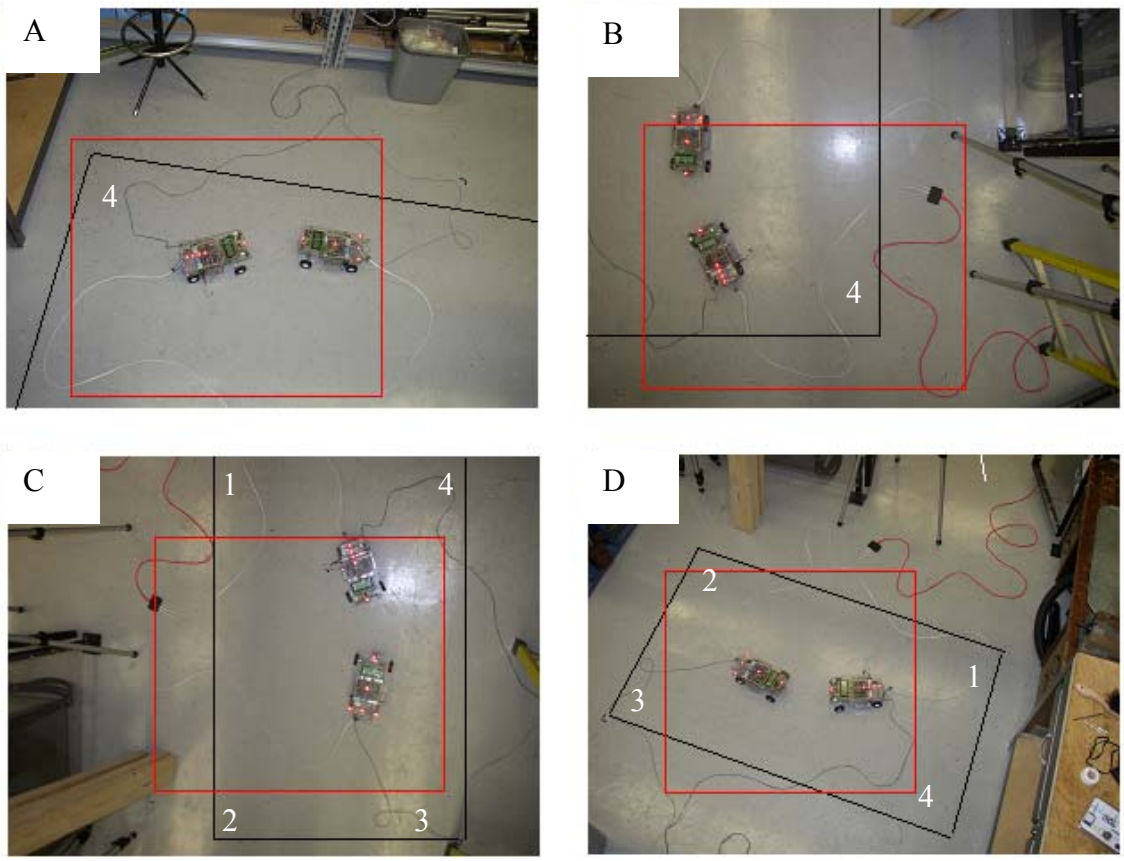

Figure 3.3. Camera views: the approximate field of each camera

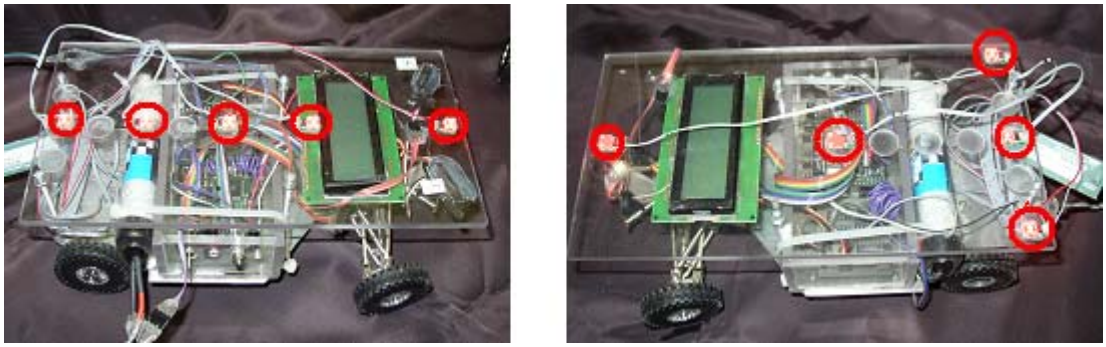

Figure 3.4. Arrangement of LEDs on the robots

Though the position measurement system gives an output of accuracy better than 5 mm, attempting to place the LEDs with such accuracy proves to be extremely difficult and therefore errors result in tracking the robot while it maneuvers. After careful placement, the accuracy of the position feedback of the front of the vehicle was still further reduced to about 7 mm. Test are run and examined with respect to this accuracy (7 mm).

3.1.3. Example Data

To better understand what the position feedback means and looks like, two examples of easily understandable patterns are given. The first maneuver to be viewed is where a single robot moves in a straight line. The second is a circle of the minimum radius of curvature of the robots.

3.1.3.1 Robot A maneuvers a straight line

Robot A is commanded to travel in a straight line with no feedback. This is helpful in determining the center position for steering (chapter 4) as well as a visual of the accuracy of the feedback to be used later. The data of the robots position is given in figure 3.5. Point A is noted by the center of the squares (green). The squares (green) are used to denote the movements of Robot A, which always travels from the right side of the plot toward the left.

There is a small patch of time where the sensor did not retrieve any position for the vehicle. These lapses are typical and the robot usually recovers rather quickly. The data given in this particular experiment took 12.5 seconds while taking data 12 times per second. This sensor rate will be standard throughout these measurements.

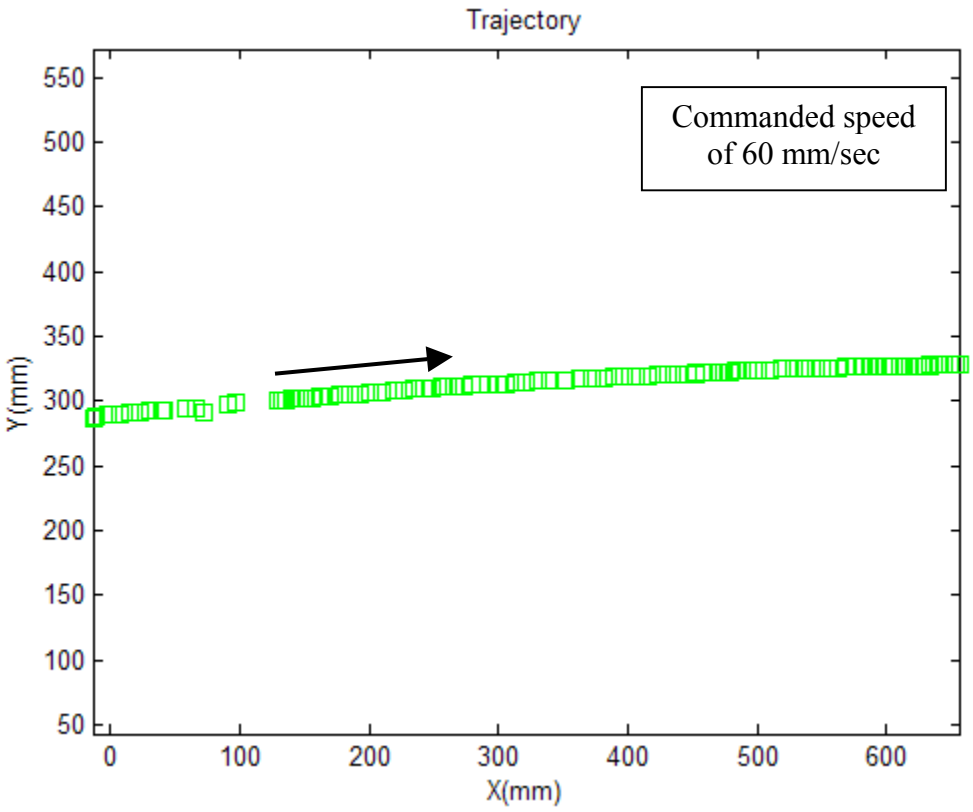

Figure 3.5. Example data of a robot traveling in a straight light

3.1.3.2 Robot B maneuvers a circle

Robot B moves along a circular trajectory that has a radius equal to the minimum turning radius of curvature for the vehicles. The experiment represented in figure 3.5 shows the movements, as the steering is held constant. The experiment lasted 25 seconds.

It is important to note that the plus signs (red) seen in figure 3.5 are used to represent Point B on Robot B. Typically Robot B would be traveling from the left side of the plot at (0,0) and moving toward the right. The robots path begins at (0,0) and continues around the circle and slightly overlapping its starting positions near the origin.

This plot is again used to exemplify that the cameras cannot always supply a position for the sensory feedback, but when it is given, the measurements are consistently

accurate within 5mm. The sparse sensory zones (circled in figure 3.6) tend to be near this circle's quadrants, limiting the workspace to about 1000mm by 1000mm. It should also be noticed from this figure that the minimum radius of curvature for Robot B is about 500 mm.

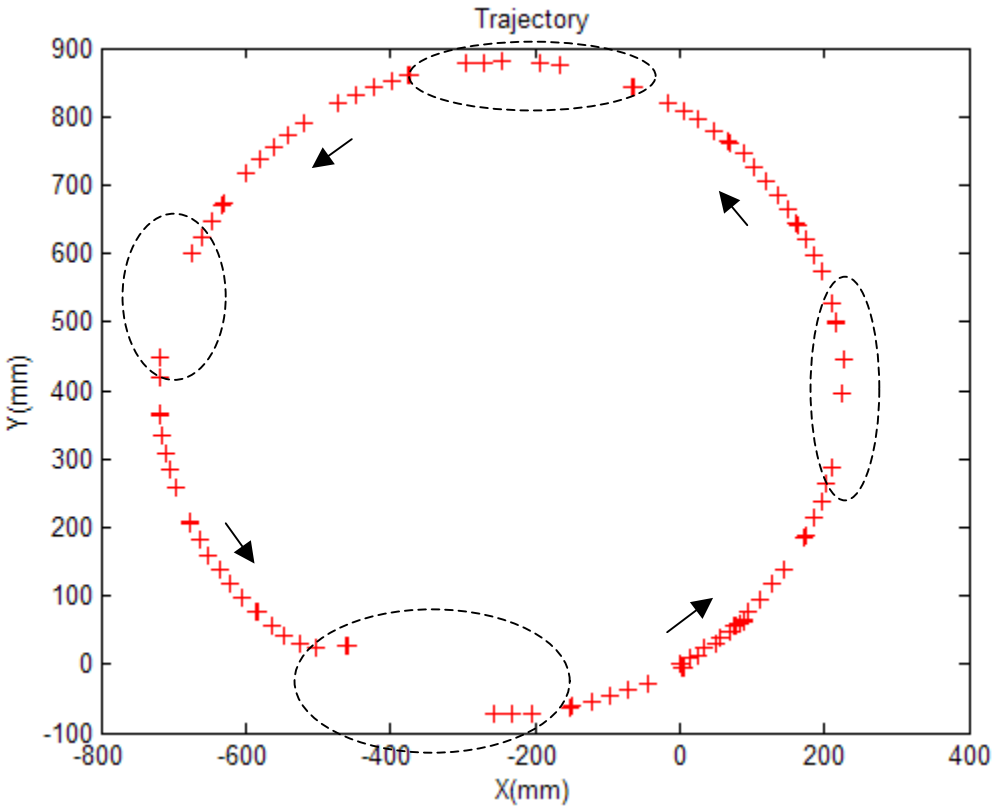

Figure 3.6. Example data of a robot traveling in a circle with the radius equal to the robots' minimum radius of curvature

3.2 Parallel Steering Mobile Robots

Robots A and B are basically identical. The information given in this section will discuss the manufacturing of one robot while its contents apply to both. Each robot consists of several parts. To have an understanding of how the path planning and docking are accomplished in this thesis it is first pertinent to understand the mechanical and electrical components of the system.

3.2.1 Mobile Platform

The base of the mobile platform is constructed from 1/8" aluminum. It is connected to a steering mechanism and drive train, both taken from a Motor Works die-cast replica of a baja racer. A box was also constructed to hold the circuitry including the LetATwork II development board for the microprocessor. Lexan was chosen because of its nonconductive characteristics. There is a top layer of the vehicle, which holds the LEDs. Likewise, it is constructed from lexan.

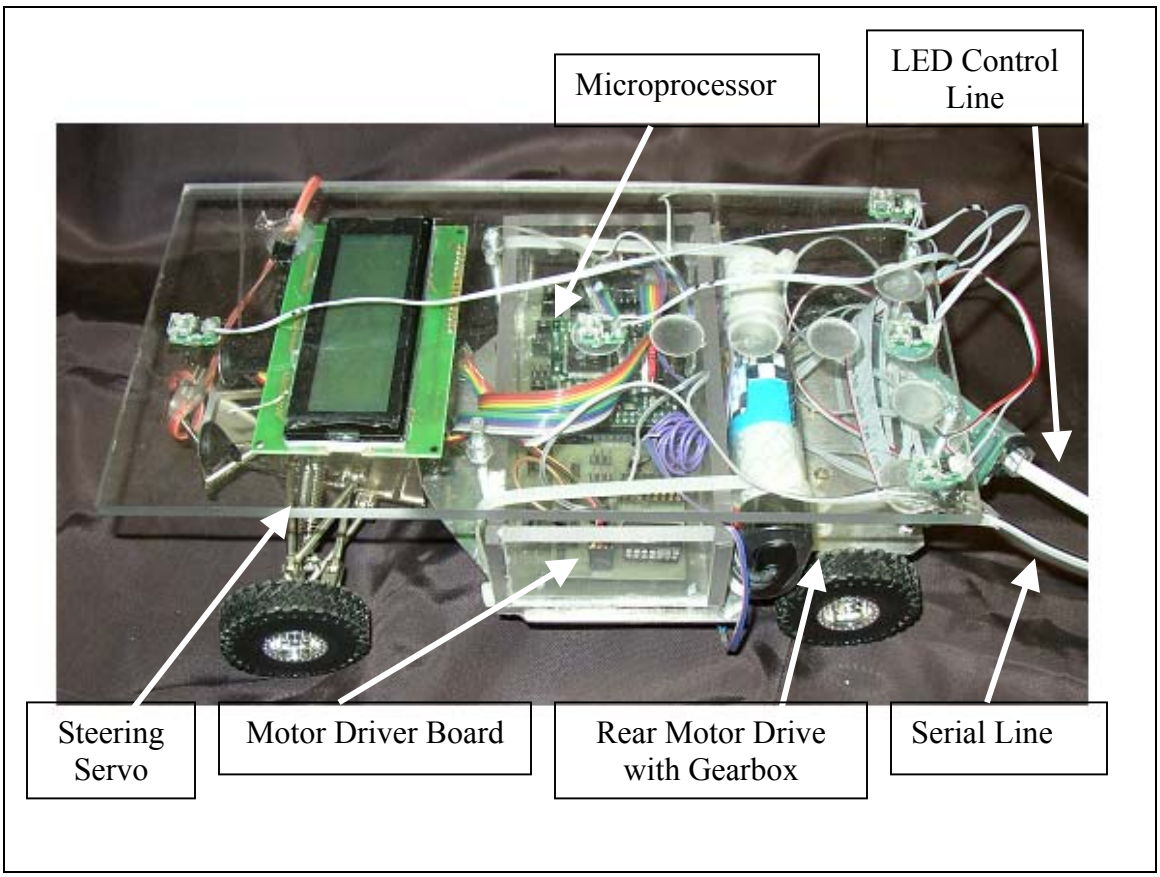

Figure 3.6 Testbed Robot

3.2.2 Motors and Drivers

There are two motors for each robot. A rear motor is used for controlling the speed of the robot. The gearbox with 203.7:1 speed reduction is made from a set of spur gears, which is actuated with a DC motor, and can be seen in figure 3.7. The speed is actuated

by a pulse width modulation (PWM) that is sent from the microprocessor to the motor driver, which allows for forward and reverse. A capacitor is added to reduce back EMF.

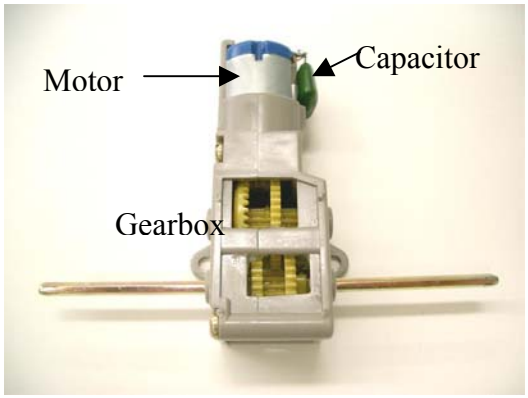

Figure 3.7. Rear drive consisting of a gearbox and motor

A servomotor is used to actuate the parallel steering. This allows for directional control with an 81-MG Hitec servomotor as seen in figure 3.8. This servomotor was chosen because of its small size and rather high torque. This servo is controlled by a PWM from the microprocessor.

Figure 3.8. Hitec servomotor (HS 81MG) used for steering

3.2.3 Microprocessor

The Atmel mega 128 is mounted on a LetATwork II board (figure 3.9) with 64 pin-outs. This board is small, and its pins are conveniently and appropriately setup. The board has two serial ports, six PWM channels, 32+ digital pins, power regulator, and

eight analog-to-digital connections. These characteristics are more than sufficient for this project. It also has 128Kb of programmable memory of which only 40 are used.

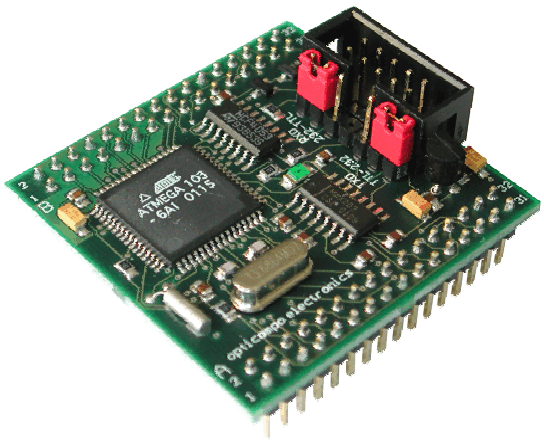

Figure 3.9. Atmel mega 128 mounted on a LetATwork II board

3.2.4 Serial Communication

Serial communication is used to transmit commands to the robots. These communications are done by connecting the serial ground and transmit pins to the corresponding ground and receive lines of the microprocessor. While many suggested latching digital communication instead of the serial line, the author found the serial port much more straightforward. Though it is not crucial, it is believed that digital communication may have allowed for faster transmissions.

CHAPTER 4
EXPERIMENTAL SETUP FOR DOCKING ALGORITHM

As the equipment (chapter 3) and theoretical background (chapter 2) of the path planning and docking have been discussed, it is further necessary to introduce the experiments that support the algorithms offered in chapter 2. Before presenting the setup and procedure of the robots' docking, calibration issues will be discussed.

4.1 Calibration of the Center Angle

When first attempting to maneuver the robots, it was noticed that the steering was skewed to the left. Much investigation went into analyzing the problem. It was finally realized that the mechanical design of the system was fine, but the servomotors were not properly centered before attaching the steering mechanisms. After several runs it was determined that a simple linear offset of the center angle would be sufficient to correct the problem. The vehicles where tested by holding the steering at the center and moving forward, which should represent a straight line.

4.1.1 Robot A

Figure 4.1 shows the original steering angle of Robot A. It is easily observed that the path is not a straight line.

The steering angle of each robot is calibrated to find the proper center angle. Because the original steering of the vehicles pulls as though the wheels make a positive angle, the steering angle is gradually incremented to the right (negative). Figure 4.2 show these five-degree steps and the angle chosen as the center. As Robot A's steering angle moved toward –15 degrees, its trajectory straightened. Though Robot A still seems

to veer as though the steering angle is set at 3 degrees, this is the most reliable center angle. When the steering angle was set lower than –15 degrees, the robot began turning to the right. This uncertainty is caused by the inaccuracies of the manufacturing of the steering mechanism.

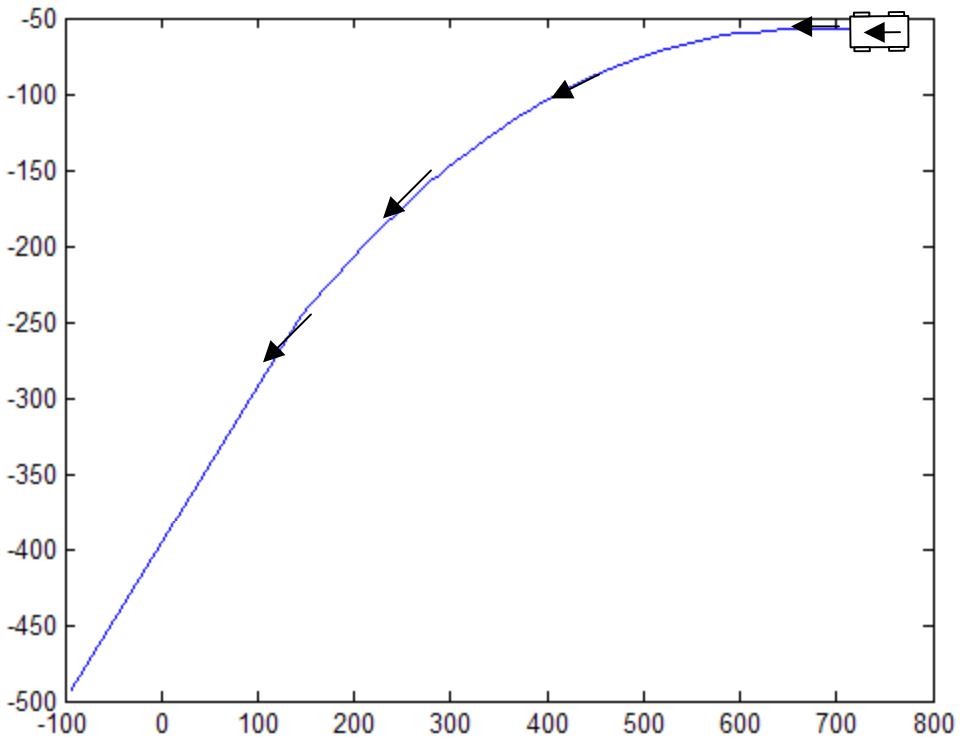

Figure 4.1. Driving vehicle in straight line at original center angle (with no correction) for Robot A

Theoretically all examples for Robot A would start in the same place; however, the robot could not be placed with such accuracy and therefore some minor differences can be notice in figure 4.2. These differences do not affect the test; they are simply visuals of the experiment.

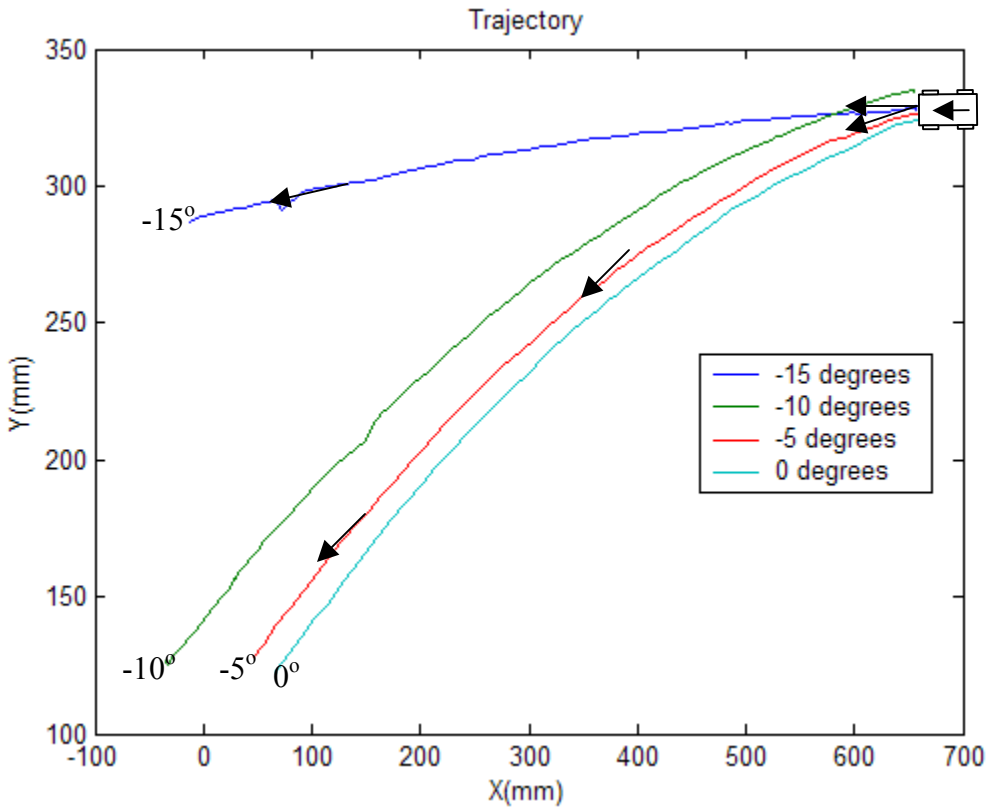

Figure 4.2. Robot A moving in a "straight line" over various steering increments

4.1.2 Robot B

Similarly, Robot B was tested for straight-line accuracy. Its results are given in figure 4.3 and are not any better than the original Robot A. As done for the previous robot, Robot B's steering is varied in 5-degree increments to the right. When the steering angle of Robot B was set at –15 degrees its trajectory became the most linear (figure 4.4). Though changing to this angle seems to revert the robot back to the left, it does show to be the closest angle to providing a straight line.

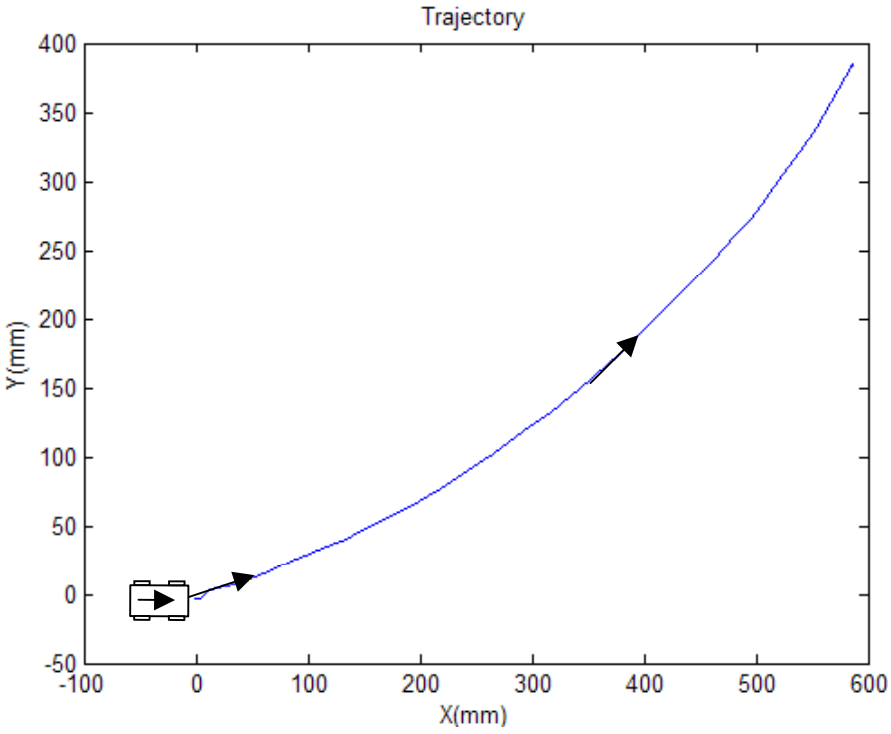

Figure 4.3. Driving vehicle in straight line at original center angle (with no correction) for Robot B

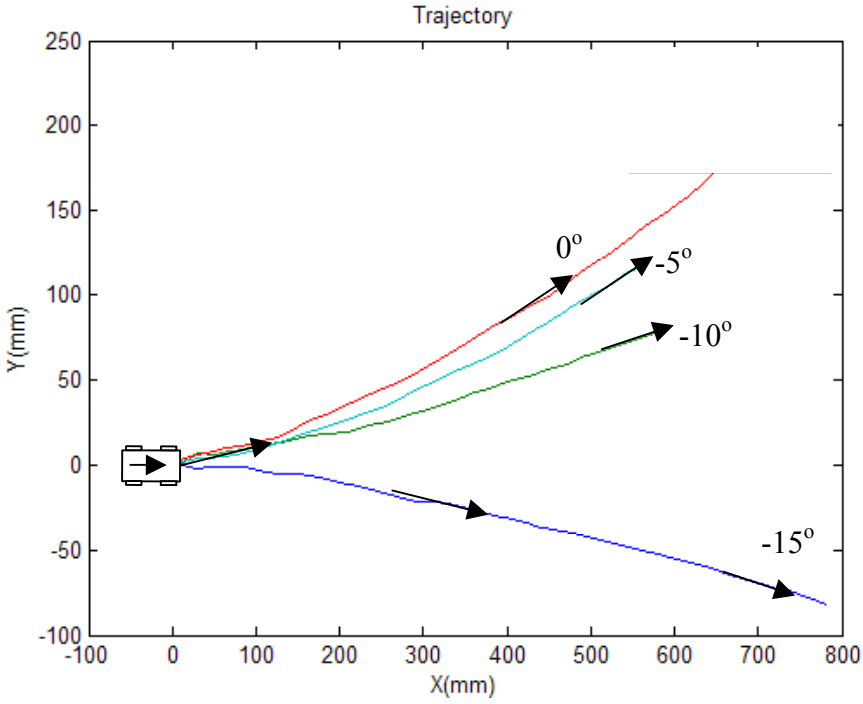

Figure 4.4. Robot B moving in a “straight line” over various steering increments

The trajectory of one of these robots, while holding the steering constant, is not consistent. This unreliability is caused by the "slop" in the mechanics of the hubs of the front wheels.

4.2 Path Planning

Now that the robots and position feedback are properly calibrated, it is pertinent to show the process and significance of each experiment. Figure 4.5, which was previously given at the end of chapter 1, is referred.

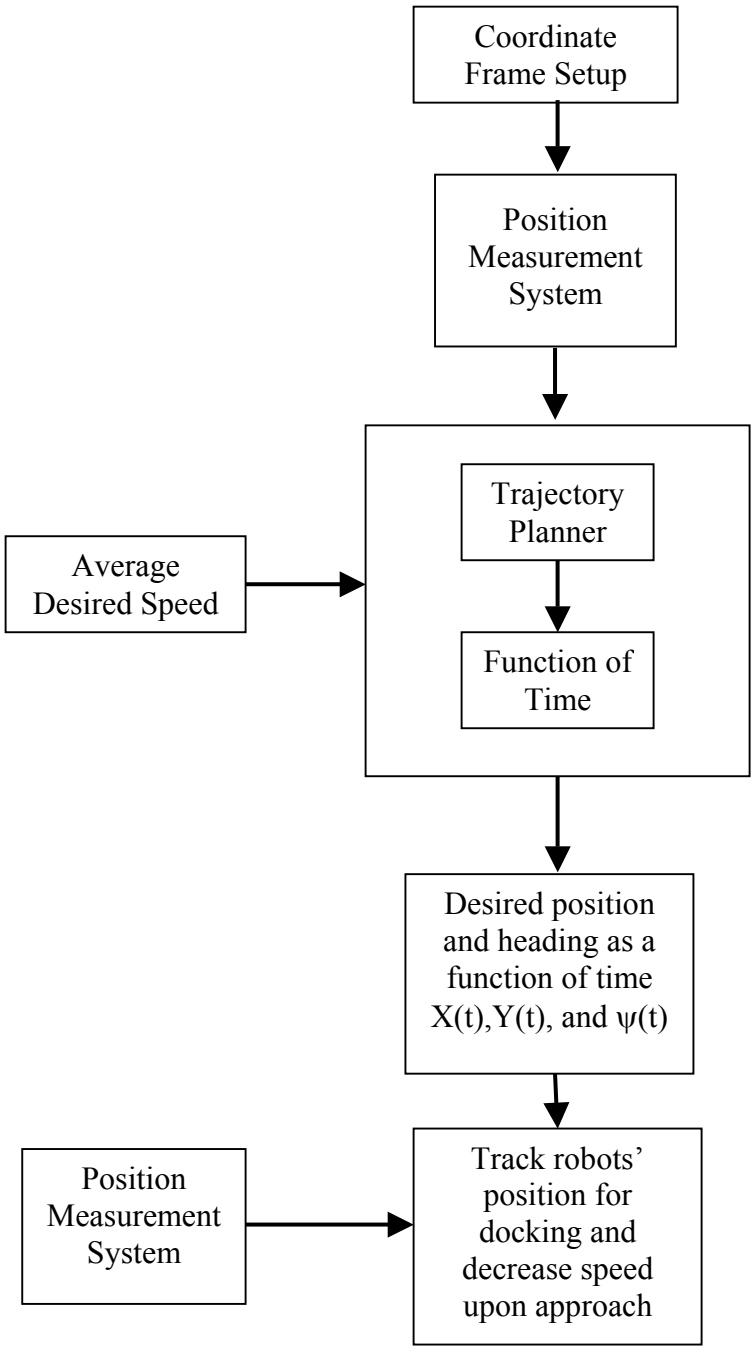

Figure 4.5. Overview of the algorithm used for path planning and tracking control

Since the coordinate frame (chapter 2), position measurement system (chapter 3), and trajectory planner (chapter 2) setup have already been detailed; there is information such as constants that need to be stated before proceeding. To determine the trajectory planner's polynomial fit for a trajectory between the robots, the algorithm is given initial end conditions defined by the robots' pose. Figure 4.6 shows a sample trajectory. It is vital to illustrate that for this algorithm Robot B's initial state defines the origin by having Point B at (0,0) and heading along the X-axis. Robot A, however, may be relatively positioned in the 1st or 4th quadrant while its heading may be defined by $90 < \psi_A < 270$.

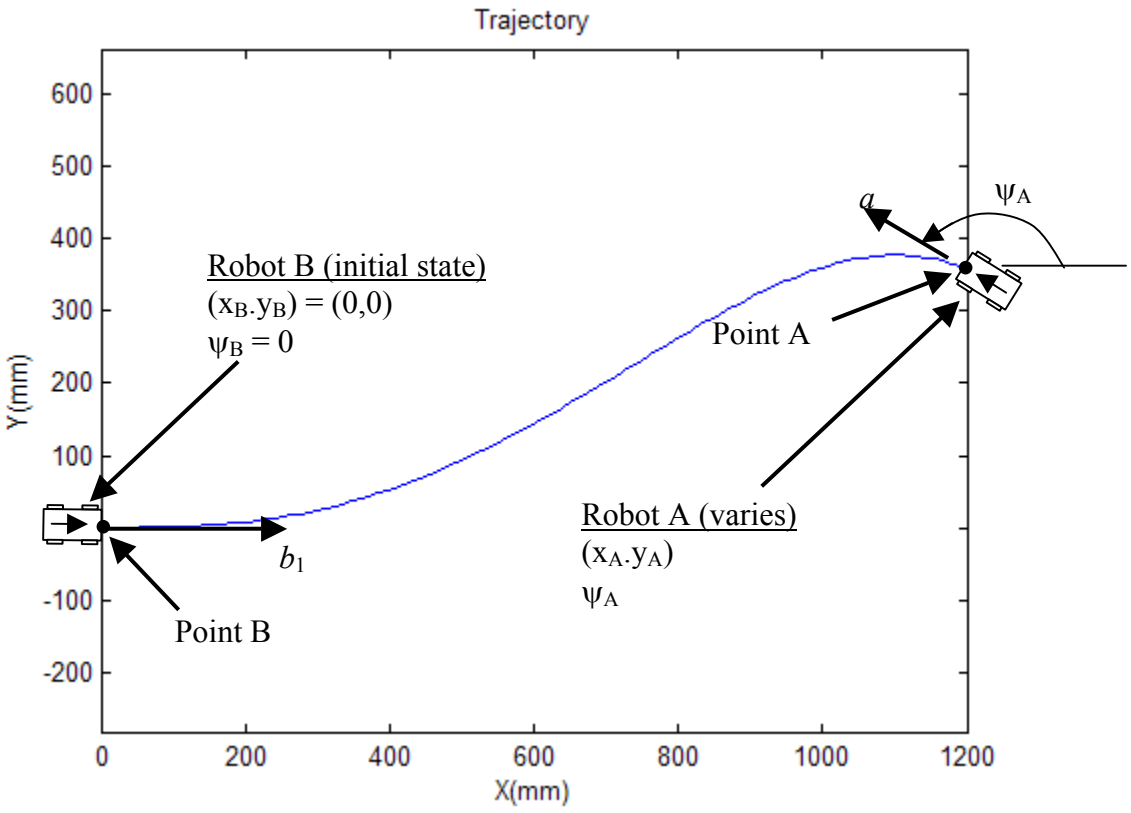

Figure 4.6. Exemplary data for constructing a 4th order polynomial trajectory

4.2.1 Maximizing the Minimum Radius of Curvature

The commanded minimum turning radius of curvature for the given testbed must be greater than 500mm. For the bisection method presented in chapter 2, α_2 is varied to maximize the radii of curvature while constraining the maximum allowed length. A sample of varying α_2 may be seen in figure 4.7. It can be observed that there are three major areas to maximize the radius of curvature (the beginning, center, and end of the path). These are the areas most in danger of not meeting the criteria because of their sharp turns. From the set of α_2 shown in figure 4.7, the minimum radius of curvature is maximized over the trajectory when $\alpha_2 = .0001$. This is in fact the only trajectory that meets the minimum radius of curvature with $\rho_{min} = 619$ mm.

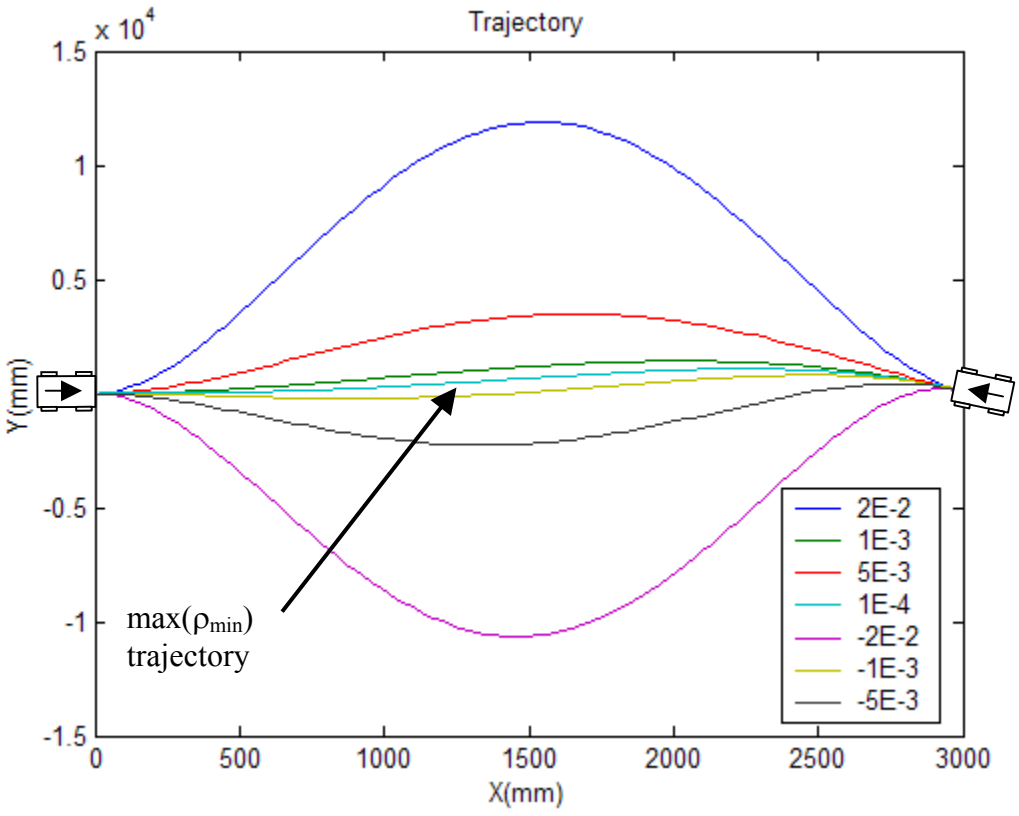

Figure 4.7. Planned paths for different α_2 values

4.2.2 Speed Controller

The speed controller was calibrated and gains tuned by commanding the robots to follow a straight line. The desired speed was set at 60 mm/sec. These results are given in chapter 5. When the vehicles are within 20 mm of their target position, the motors are turned off and with no power the robots quickly come to a stop.

4.2.3 Position and Heading Controllers

The position and heading controllers are designed by analyzing each controller separately and then combining them for a superior control scheme. Because these robots do not have models, the values for the control constants of the PID controller were determined experimentally. Several methods such as Zeigler-Nichols method were examined but found incompatible due to high non-linearity and machining inaccuracies, therefore, the constants were found by educated trial and error. For each scenario, a proportional controller is first implemented. When the rise time is near a desired amount, which is relative, a derivative controller is added to the system to damp the oscillations that are sure to have crept in. If there is any resulting steady state error, then an integral control is added to the system. After each of the controllers, position and heading, have been analyzed independently they are combined and the gains are fine-tuned. Adjustments are then made to account for the combination. After this has been done for both robots separately, the vehicles are allowed to travel along the trajectory until they meet.

CHAPTER 5
RESULTS FOR DOCKING ALGORITHM

Since the background (chapter 1), theory (chapter 2), and experimental setup (chapter 4) for the path planning and docking have been given, a basic understanding is now had for the results of this research development and investigation. It is important to analyze Robots A and B individually since their paths are distinct and their mechanics are slightly different because of error in construction. In this chapter a discussion is given for each of the controllers used. The speed, position, and heading are each maintained with a proportional derivative (PD) controller. Each section discusses the controller for each robot and its effects. The later part of the chapter presents the results of both robots traveling along the trajectory and docking.

5.1 Speed Control

To experimentally test the control of the speed of each robot, PID controllers were tested. The purpose of these controllers was to keep the vehicle moving along the trajectory in concordance with its predetermined function of time. If the robots were ahead of their intended position, instead of backing up, they were commanded to stop and wait for the commanded position to catch up.

Figure 5.1 shows an example of tracking Robot B along the predetermined trajectory. There are a few key issues with tracking that appear in this example. The first noticeable problem in this figure is the lack of data near the left side. This is caused by a loss of LEDs where the cameras could not detect the pieces of data needed for tracking the position or heading of Robot B. This sensor malfunction (loss of data) will be briefly

discussed in section 5.6. While speed is difficult to see in the figure, the areas with sparse data points represent swift movements by the robot. The paused areas show where the robot was traveling ahead of its commanded position and was instructed to stop. The cluster of data points shows that the robot remained in the same spot for a time. These stops and starts are the results of a poor speed controller.

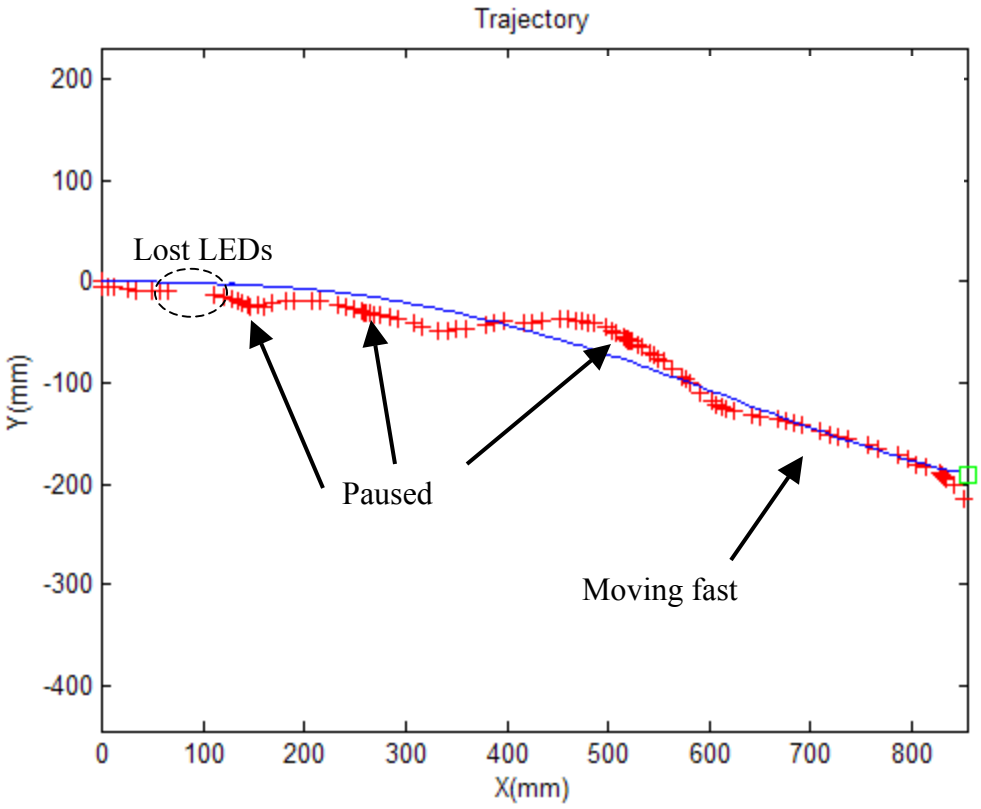

Figure 5.1. Example data for a poor controller that has some stopping and starting

The robot is commanded to follow the X(t) curve as close as possible. If the robot does not maintain a rigid performance where the distance between the actual and commanded is minimal then the robot will miss or cut curvatures in the trajectory. For example, in figure 5.2, Robot A delays too long at the beginning of the experiment and causes it to miss most of the predetermined path. Because the robot was not strictly

tracking the path, as it attempts to correct the steering, it cuts the first commanded curvature and undershoots the second.

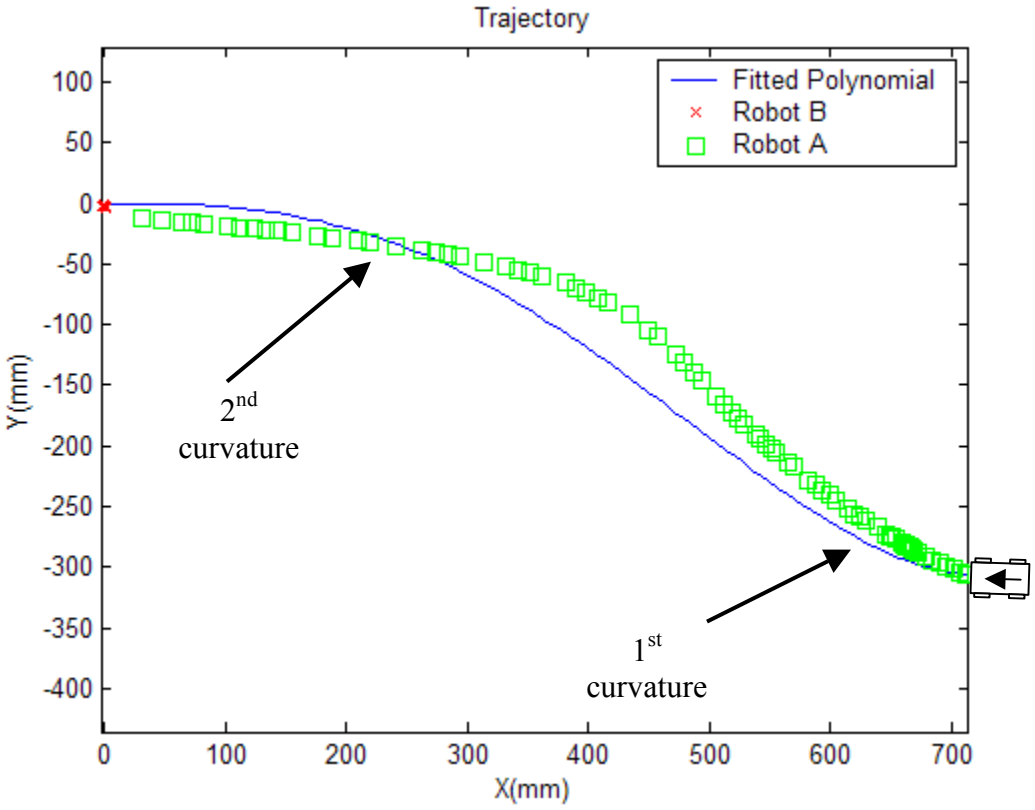

Figure 5.2. Robot A moves too slowly causing it to veer from its course

It can also be seen that there is a small offset between the endpoint of the robot's movement and its target. This is caused by the speed controller, which was set to terminate the movements at a distance of 20mm for each robot. For most of the trials, the sensor data is taken slow enough such that the robot actually ends closer or beyond its goal, which is the exact reason for the premature termination. For this experiment, the halting of the movement worked as prescribed, but left a final offset. This problem cannot be addressed in this thesis, but its study is recommended for future work.

5.1.1 Speed Control for Robot A

The following set of experiments were conducted using Robots A and B independently. To create the trajectory as a function of time a desired speed was set constant at 60mm/sec. Robot A was allowed to travel about 700mm along a commanded straight path. A simple steering controller, which was used consistently throughout the tuning of the speed controller gains, was used to correct the position of the robot as it traveled along the curve. It is important to understand that the trajectories for the tuning process are straight lines.

In the next several figures (e.g. figure 5.3) it is important to notice a few key aspects of the graphs. The left half of the figures show the commanded and actual X position plotted against time. The green line, actual, should always lag behind the commanded. Any flat (horizontal) portions of the curve indicates the robot pausing while awaiting the error signal of the position to be positive, X(t)>x(t), also called lag. The right hand plot shows the deviation of the robot from the commanded position. This includes an absolute distance of the X and Y errors,

$$R = \sqrt{(X - x)^2 + (Y - y)^2} \tag{5.1}$$

where (X, Y) is the command position and (x,y) is the actual position.

The speed controller was determined by having the robots travel in a straight line as detailed in chapter 4. First a proportional controller is constructed and confirmed to work unsatisfactorily for Robot A. While there is a rather smooth transition between stopping and starting, the pauses of the robot are considered too long (figure 5.3). In some cases pauses caused the position error to rapidly grow (e.g. the major pause in figure 5.3 resulted in the maximum error, R). There are four types of reactions. The first, pausing,

has already been mentioned. This occurs when the robot is ahead of its commanded position. The second, sluggish acceleration, occurs after the robot has paused and is beginning to move but cannot accelerate quickly enough to follow the curve. The third, delaying, usually occurs only at the beginning of the path or after a pause when the error has not grown large enough to command motion from the drive motor. The final reaction is most desirable, when the robot has a constant lag behind its commanded position. By definition this occurs when the speed oscillations are minimal. This lag, of course, is desired to be as small as possible.

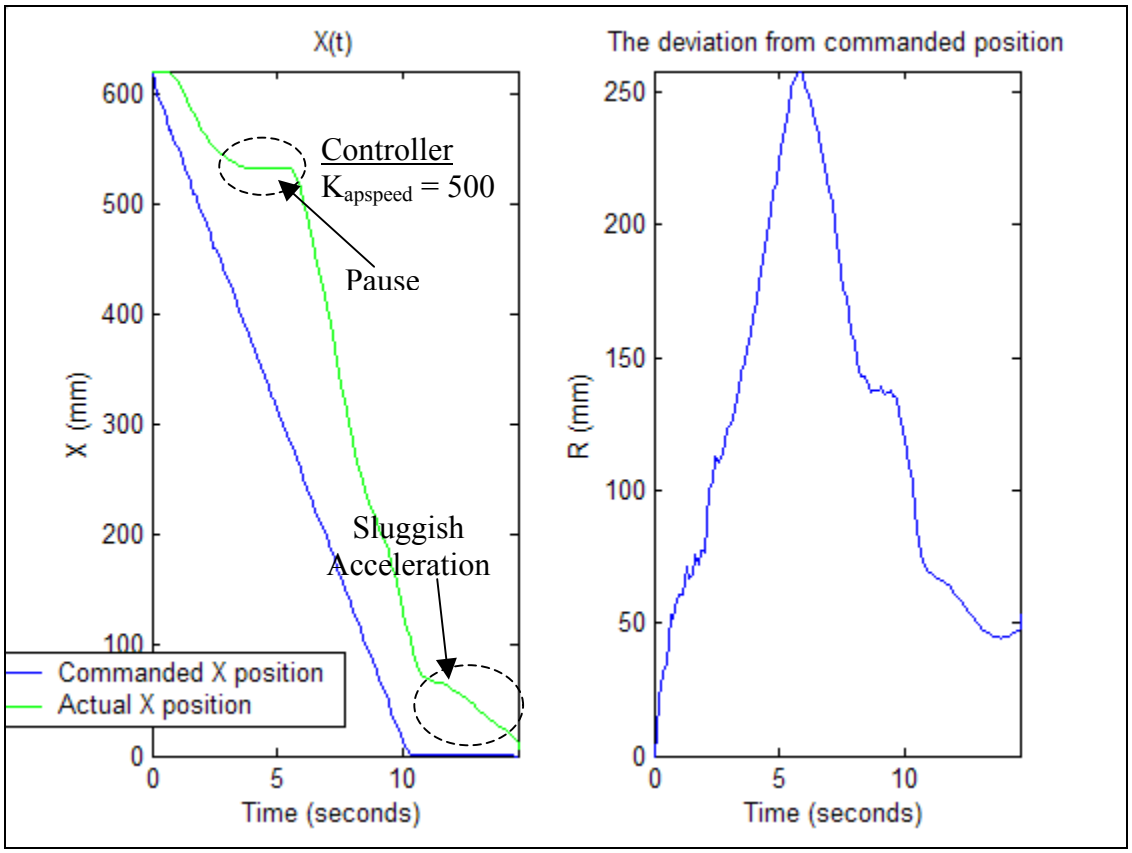

Figure 5.3. Proportional speed controller for Robot A ($K_{apspeed}$ = 500)

When the data on the left portion, X(t), of figure 5.3 is a straight, constant slope, the robot is traveling at a constant speed. The line that is more vertical is the traveling faster. Thus after the pause (figure 5.3) the robot is traveling faster than the desired 60

mm/sec to regain it commanded position. Likewise, when a set of data is horizontal on the right potion, deviation R(t), the vehicle is traveling at the desired speed, which means the error is staying the same.

Next, the proportional control constant value was increased. Damping was used to control the oscillations about the desired speed by adding a derivative controller to the already implemented proportional. This greatly improved Robot A's performance (figure 5.4). While the majority of the experiment has a constant lag near 110mm, the transitions are extremely smooth with no stopping or pausing until the course is traversed. Furthermore, the consistent lag implies a constant speed, which is highly desirable. The maximum lag nearly reaches 130 mm. This is about half the lag of the experiment described by the proportional (figure 5.3). Some of the drastic variations in the deviation plot are caused by errors in the *y* position. Since the experiment is solely focused on the linear motion of the robot, the *y* discrepancies are ignored where there are sharp changes less than 10 mm. The changes could also be produced by the accuracy of the position feedback mentioned in section 3.1.2.

A proportional integral derivative (PID) controller was discussed for implementation and found that the integral did not benefit the system. The major advantage of adding an integral controller in classical controls is to improved steady-state error. Because the steady state error was already sufficiently small, the addition of the integral was unnoticed and therefore considered unnecessary.

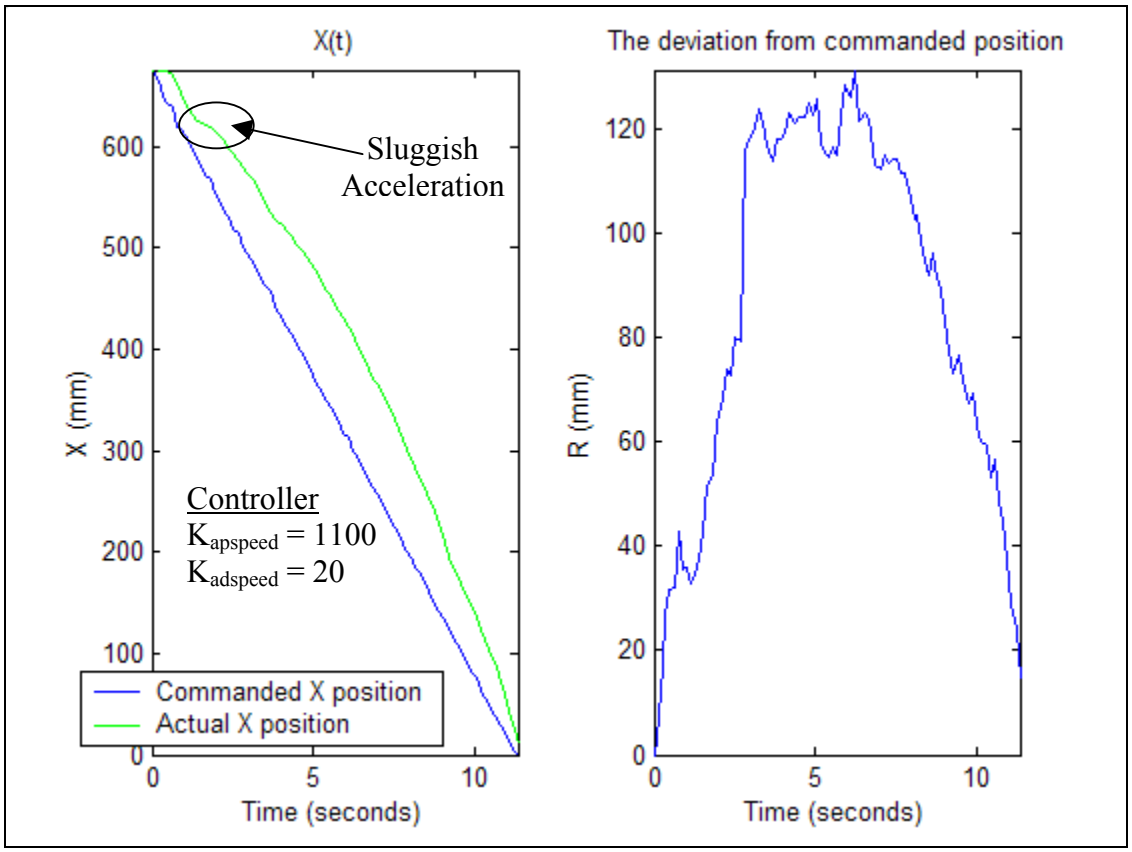

Figure 5.4. Proportional derivative speed controller for Robot A ($K_{apspeed}$=1100 $K_{adspeed}$ = 20 Robot A)

5.1.2 Speed Control for Robot B

Similarly, the speed of Robot B was analyzed and gains selected for its PD controller. Figure 5.4 shows a controller with a significant amount of the derivative controller. This unfortunately caused the system to lag too much. This controller, however, already responds much better than the controller for Robot A. It should also be noticed that because Robot B is traveling opposite of Robot A, its control constants have an opposite sign.

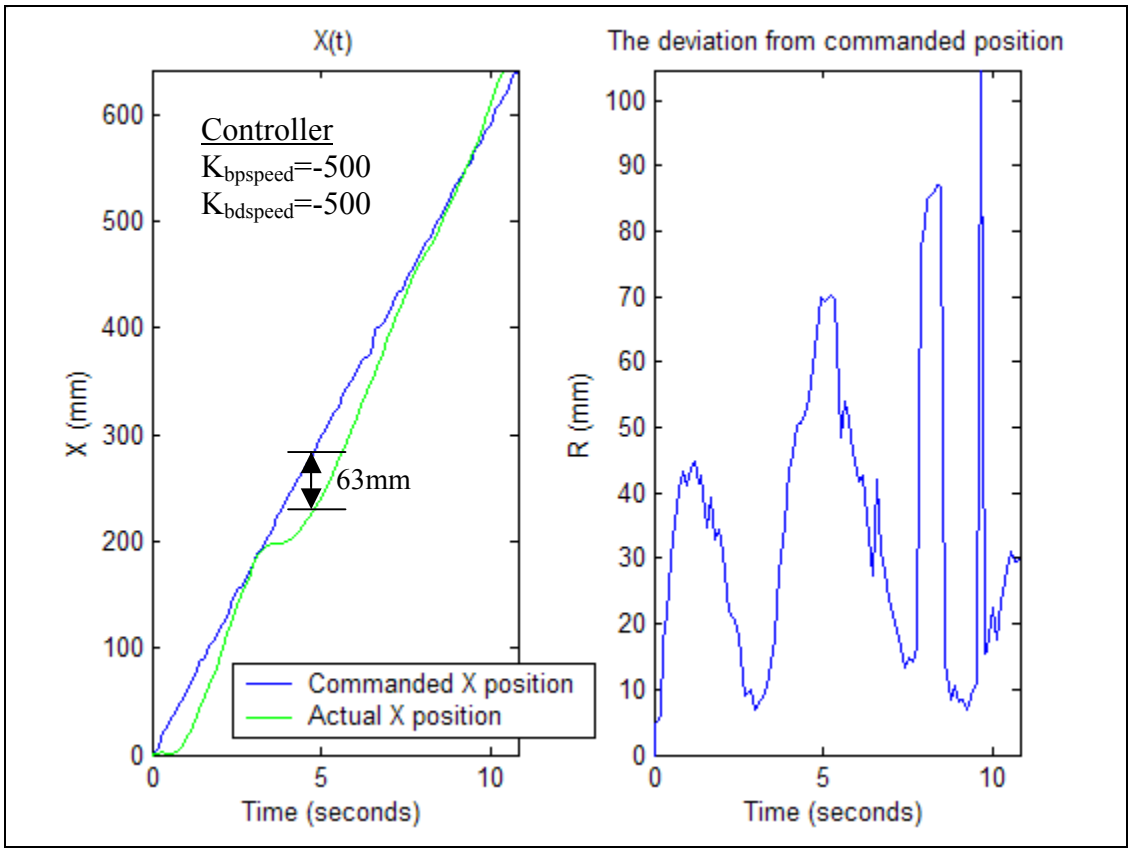

Figure 5.5. PD speed controller for Robot B with equal control constants ($K_{bpspeed}$ = -500 $K_{bdspeed}$=-500 Robot B)

The derivative portion of the controller was decreased and the proportional increased to yield a quicker response of the robot (figure 5.5). The greatest lag in the system is 63mm. In light of the previous discussion of neglecting drastic changes of R that are less than 10 mm, the jumps of 80 and 90 mm should be easily noticed and considered as a result of changes in the *y* direction. Though the robot was at its commanded X(t) position, its *y* direction had between 20 and 50 mm of error. Similar errors of R from *y* are also seen in figure 5.6.

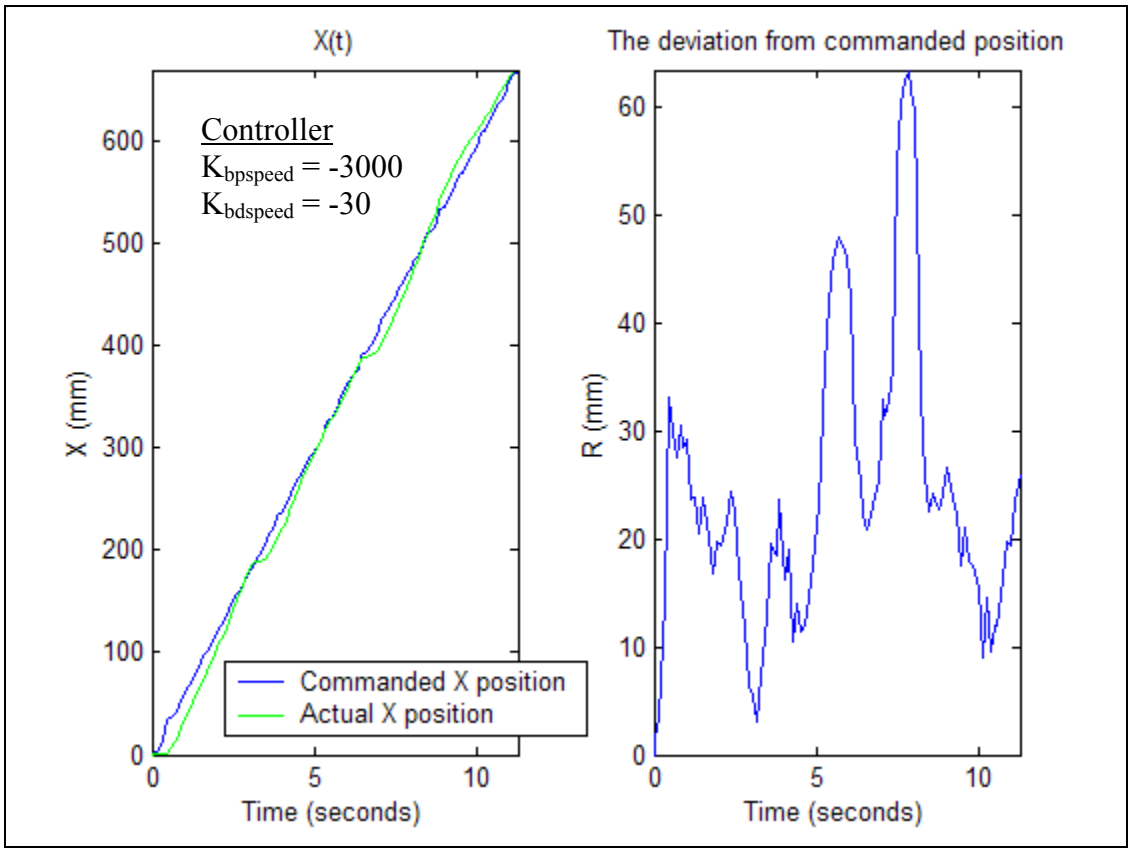

Figure 5.6. Proportional derivative speed controller for Robot B ($K_{bpspeed}$ = -3000 $K_{bdspeed}$ = -30)

The most successful controller in maintaining a tolerance on the path is given in figure 5.6. The largest lag error in X(t) is 33 mm and is caused by the delay at the beginning of the maneuver.

After seeing such large variations between the controlled responses of the vehicles, the difference in the rear motors was strongly distinguished. It is believed that because these motors were cheap (toy DC motors) their inefficiency and lack of reliability caused these major differences. In spite of the changes between the different motors, the speed controllers for each robot work well and are consistent.

After showing the importance of the speed controller, the final control constants for each robot are show in Table 5.2.

Table 5.1. Performance results of various control constants on the speed controller of Robot B

Kp	Kd	Time	Length	Average Desired Speed (mm/s)	Average Actual Speed (mm/s)	Largest Lag (mm)	No. of Sluggish Accelerations
-1000	0	10.4	675	60	64.90	88	2
-1500	0	11.1	653	60	58.83	140	1
-2000	0	9.2	644	60	70.00	99	3
-2500	0	10.5	665	60	63.33	92	3
-3000	0	10.3	645	60	62.62	97	3
-2000	-10	11.3	697	60	61.68	184	2
-2000	-20	10.8	712	60	65.93	113	3
-2000	-25	11.2	677	60	60.45	147	2
-2000	-30	10.5	674	60	64.19	90	3
-3000	-30	9.5	646	60	68.00	62	2
-2500	-30	9.9	660	60	66.67	78	2

Table 5.2. Control Constants used to regulate speed of each robot

	Robot A	Robot B
Proportional (K_p)	1100	-3000
Derivative (K_d)	20	-30
Maximum Lag (mm)	128	33
# of Pauses or Delays	1	2

5.2 Position Control

Now that the speed controller has been thoroughly discussed, it is appropriate to examine the added control for correcting the position of the robots as they travel along the path. The speed controllers previously introduced are used throughout the remainder of this thesis. The next controller that is implemented is that for regulating the position of the robot. A short analysis of how accurately they followed different paths is discussed with a final recommendation for a control constant to minimize the position error deviations from the predetermined path. This section will detail the derivation of the chosen controllers for each robot.

5.2.1 Position Control for Robot A

In this set of the experiments, Robot B remains motionless while the position of Robot A is considered as it travels from the "right" and is represented by squares (green) in the figures below. Given by the path planner, the equation of the polynomial and the distance, x, as a function of time, the position of a point on the robot can be analyzed in time. As stated in chapter 2, it is important to track the x and y position of the point that makes up the center of the front edge (point A) of Robot A. This point is not specifically tracked, but is virtually tracked knowing its relative position to other features (LEDs) on the robot.

The first controller used is a proportional controller. After several trial-and-error runs an acceptable response was obtained, shown in figure 5.7. Its motion follows the trajectory well but continually overshoots the desired position causing oscillations about the nominal trajectory. Initially, the robot delayed in starting its motion. This caused the desired position on the trajectory to be a point ahead of its current position, which it should; however, it continually overcorrected as it approached its desired position and was required to make a sharp turn back into the desired trajectory.

The overshoots along the maneuver represented in figure 5.7 can be minimized by adding a derivative controller. The derivative acts as an oscillatory damper and slows the large reactions of the steering. Similar to figure 5.7, Robot A in figure 5.8 has a slow start and reacts quickly to catch up; however, in this instance when it turns back into the desired path the vehicle does not overcorrect, but makes a smooth transition back on course.

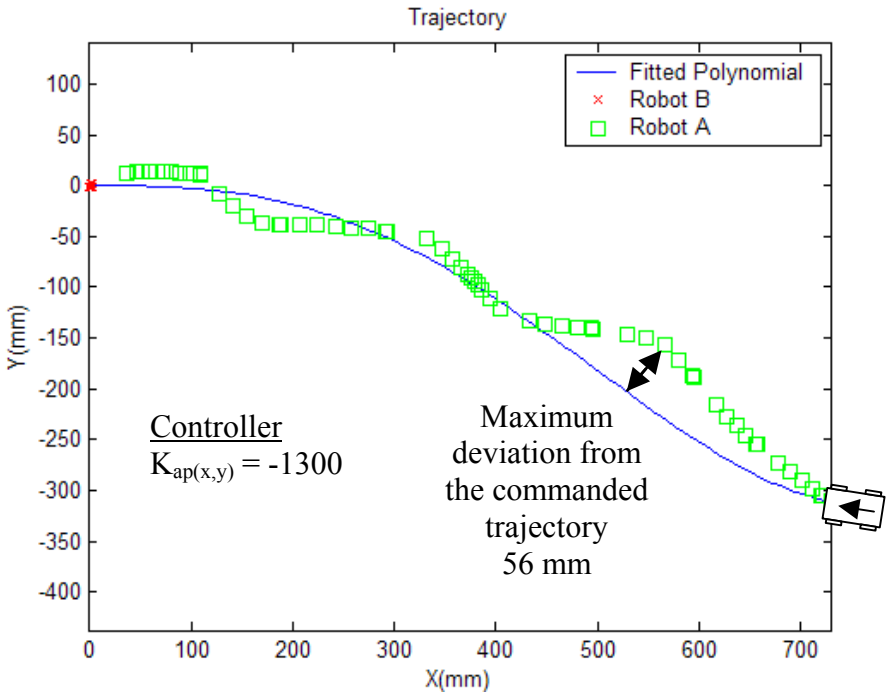

Figure 5.7. Maneuver of Robot A with a proportional controller ($K_{ap(x,y)}$ = -1300) on the position

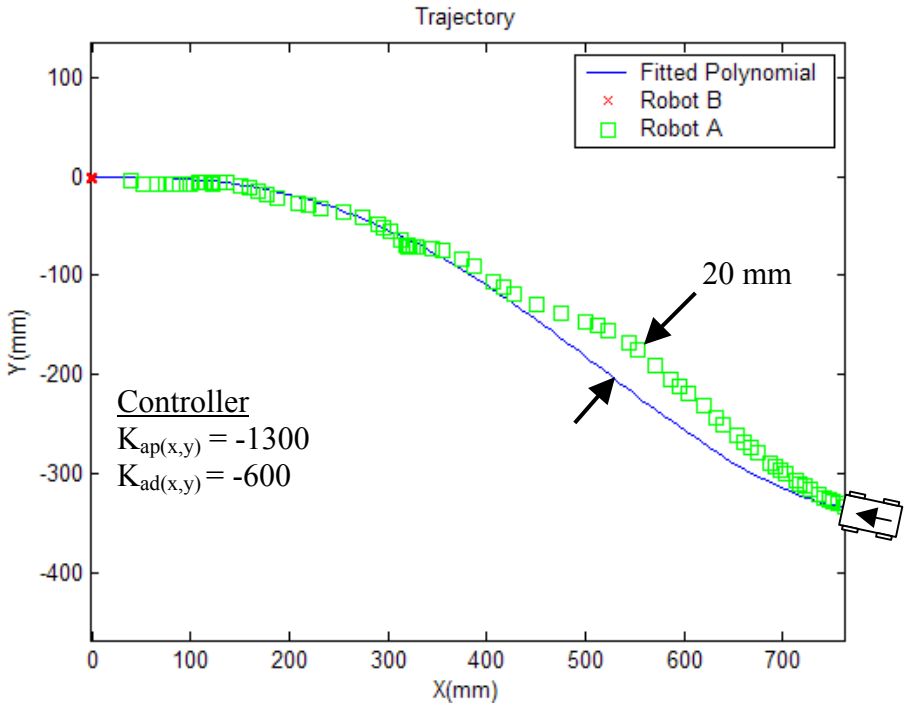

Figure 5.8. Maneuver of Robot A with a proportional derivative controller ($K_{ap(x,y)}$ = -1300 and $K_{ad(x,y)}$ = -600) on the position

The proportional constant used for this experiment was –1300 while the derivative portion was nearly half at –600. This is a significant addition of damping. The values are reiterated in table 5.3. The deviation from the desired course is measured as the absolute difference between y(x) and Y(X).

Table 5.3. Implementation of proportional and proportional derivative controllers on Robot A

	Control Constants	Maximum Deviation
Proportional only	$K_{ap(x,y)} = -1300$	56 mm
Proportional Derivative	$K_{ap(x,y)} = -1300$ $K_{ad(x,y)} = -600$	20 mm

5.2.2 Position Control for Robot B

Similar to the experiment for Robot B's trajectory controller, a controller was constructed for Robot A. The first controller implemented was a proportional controller. An acceptable controller was found ($K_{bp(x,y)}$=1500). As with most simple proportional controller, good handling of the vehicle was not quite achieved. The results for this implementation are given in figure 5.9. Robot B is distinguished by red x's, which travel from the left at (0,0) to the right. There were some oscillatory deviations from the path that caused the robot to approach its docking port at the wrong angle.

As expected, the largest deviations occur immediately following a curve. If there are any oscillations on a straight segment of the path, the robot can almost always turn back and recover. However, when the robot encounters any oscillations on a curve the curvature in the curve is not taken into consideration by the controller and therefore the error often grows. The controller is only concerned with an (x,y) error. Even small over-shooting on a curve can result in a large difference on the opposite side of the curve. Though it is a complicated issue, a controller that is robust to curves is needed for all path planning.

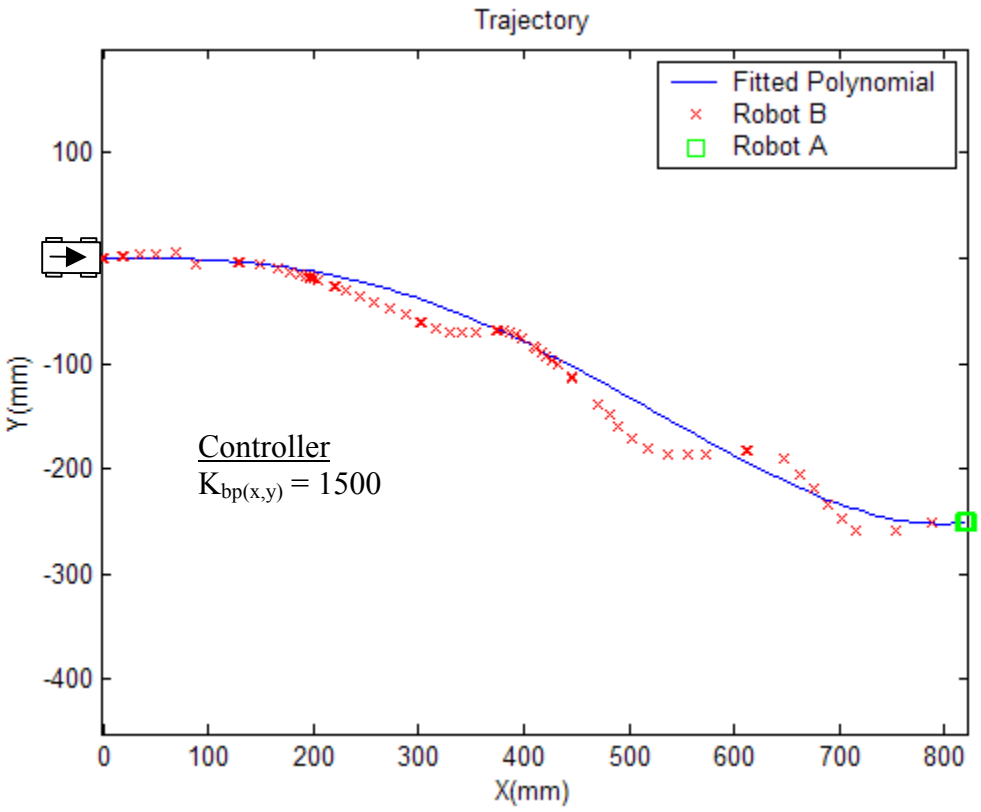

Figure 5.9. Maneuver of Robot B with a proportional controller ($K_{bp(x,y)}$=1500) on the position

As previously done for Robot B, a derivative controller was implemented to correct the tracking of Robot A. The misalignment was fixed and the vehicle followed the trajectory tighter with the improved controller. This enhancement can be viewed in figure 5.10. The robot nearly misses the correct heading at the end of the run because of oscillations about the final curve. This second curve is much tighter than the first because of the path planner.

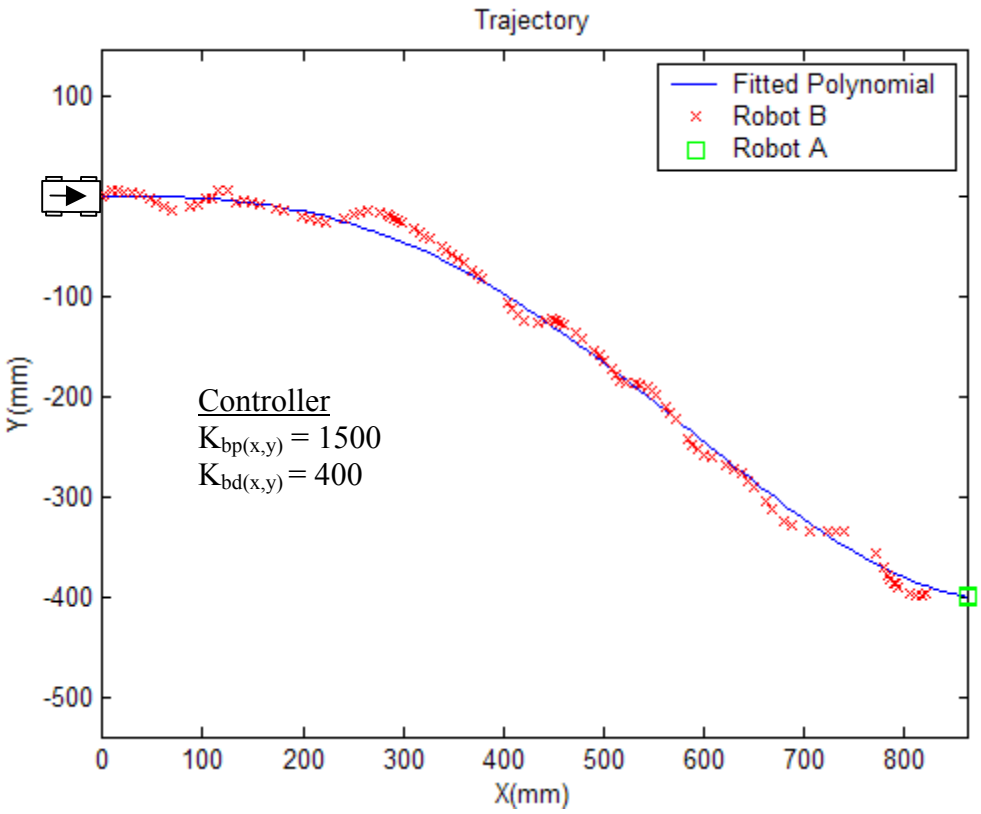

Figure 5.10. Maneuver of Robot B with a proportional derivative controller ($K_{bp(x,y)}$ = 1500 and $K_{bd(x,y)}$ = 400) on the position

After the final controller was tested, its results were tabulated and recorded. Table 5.4 displays the values for the control constants and the deviation from the desired movements of the robot. It is important to again notice how the large damping greatly enhanced the controlling of these vehicles. The change in deviations from applying the derivative controller reduced the deviation by 57.5%, which is a significant enhancement to the controller. The robots performance was again poor around the final curvature, but with added features to the controller such as heading control, the curvature does not greatly affect the robots as it did here assuming that the criterion for minimum turning radius of curvature is met. This is discussed in the next section.

Table 5.4. Implementation of proportional and proportional derivative controllers on Robot B

	Control Constants	Maximum Deviation
Proportional only	$K_{bp(x,y)} = 1500$	40 mm
Proportional Derivative	$K_{bp(x,y)} = 1500$ $K_{bd(x,y)} = 400$	17 mm

While there were great differences in the results of the speed controllers, the steering controllers for position correction of each robot was quite similar (table 5.5). The maximum deviation of the robots was consistently less than 20mm.

Table 5.5. Performance results of proportional derivative controllers for each robot

	Robot A	Robot B
Proportional (K_p)	-1300	1500
Derivative (K_d)	-600	400
Maximum deviation (mm)	20	17
Final Position Error (mm)	2	4

5.3 Heading Control

The effects of using a heading controller by itself are analyzed in this section. The speed controller from the section 5.1 is used. There is no position control implemented and a new heading controller is applied. Just as important as positioning the robots correctly are their headings as they approach and dock with their target. Knowing the derivative of the polynomial and x(t) allows the desired slope for the robot to be calculated. This desired orientation is compared with the current direction and an error signal is produced. With a proportional and proportional derivative the robots are routed to their targets.

5.3.1 Heading Control for Robot A

A heading controller was developed and employed for Robot A. Figure 5.11 shows the results of such application. Though the robot does not follow the path as intended, it

does do a fairly good job of holding the proper heading. The initial slope up and the final approach to the target are correct. Having a controller only on the heading tends to exaggerate the desired path. A final proportional value was chosen as $K_{ap\psi} = 2000$ and a derivative value as $K_{ad\psi} = 1000$.

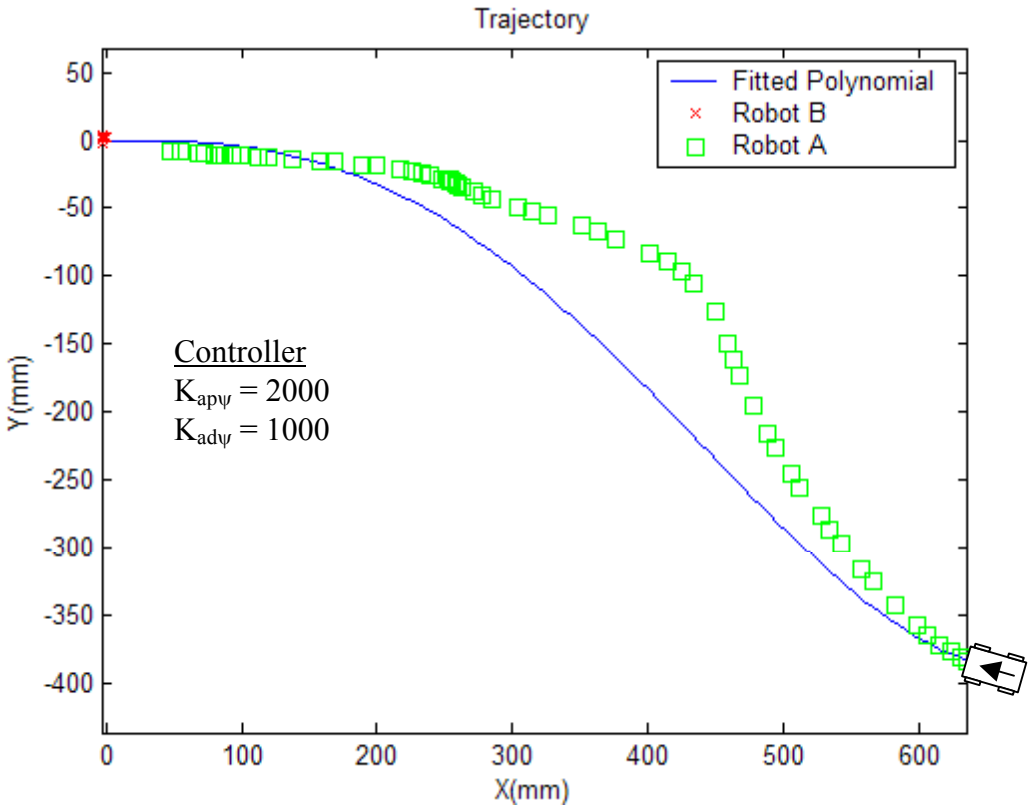

Figure 5.11. Maneuver of Robot A with a proportional controller on the heading ($K_{ap\psi} = 2000$ $K_{ad\psi} = 1000$)

5.3.2 Heading Control for Robot B

A heading controller was also considered for Robot B. Similar to Robot A, the vehicle does not following the position of the planned course well; however, it mimics the slope well. Figure 5.12 shows the position of Point B on the vehicle. By visually following the change in the data points, the heading of the robot can be seen. Unlike the maneuvers for the position controllers, the paths for the directional controlled robots are

extremely smooth with no overshoots. A final proportional value was chosen as $K_{bp\psi}$ = 2000 and a derivative value as $K_{bd\psi}$ = 1000.

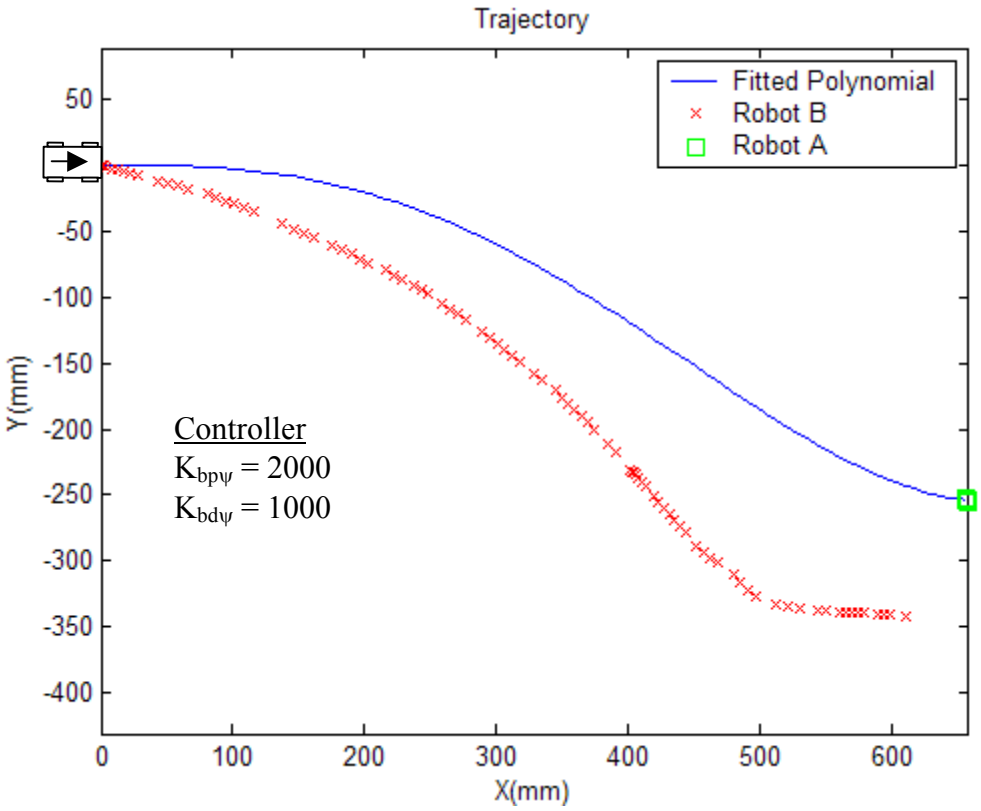

Figure 5.12. Maneuver of Robot B with a proportional controller on the heading ($K_{bp\psi}$ =2000 $K_{bd\psi}$ = 1000)

5.4 Position and Heading Control

As seen in the previous sections heading and position controllers are not sufficient by themselves. A combination of the two is necessary to achieved proper tracking and orientation. It was observed that the heading controller acts like a damper. This occurs when the position error may be oscillatory about the nominal path, but with the heading controller added to the system, there is a constant drive pushing the vehicle parallel to the curve. A greater emphasis was placed on the position errors of the robot to ensure their arrival at their target location. For the most part the position controller remained the same for the robots, while the heading emphasis was decreased.

5.4.1 Position and Heading Control for Robot A

The position and heading of Robot A were controlled first. Since adding these two controllers is not straightforward there were many adjustments to them so they could work in conjunction. The function of the controllers together had a tendency for the position controller to dominate when the error from the position became too large. As it went back to zero, the heading controller dominated the control action. This balance was a careful integration.

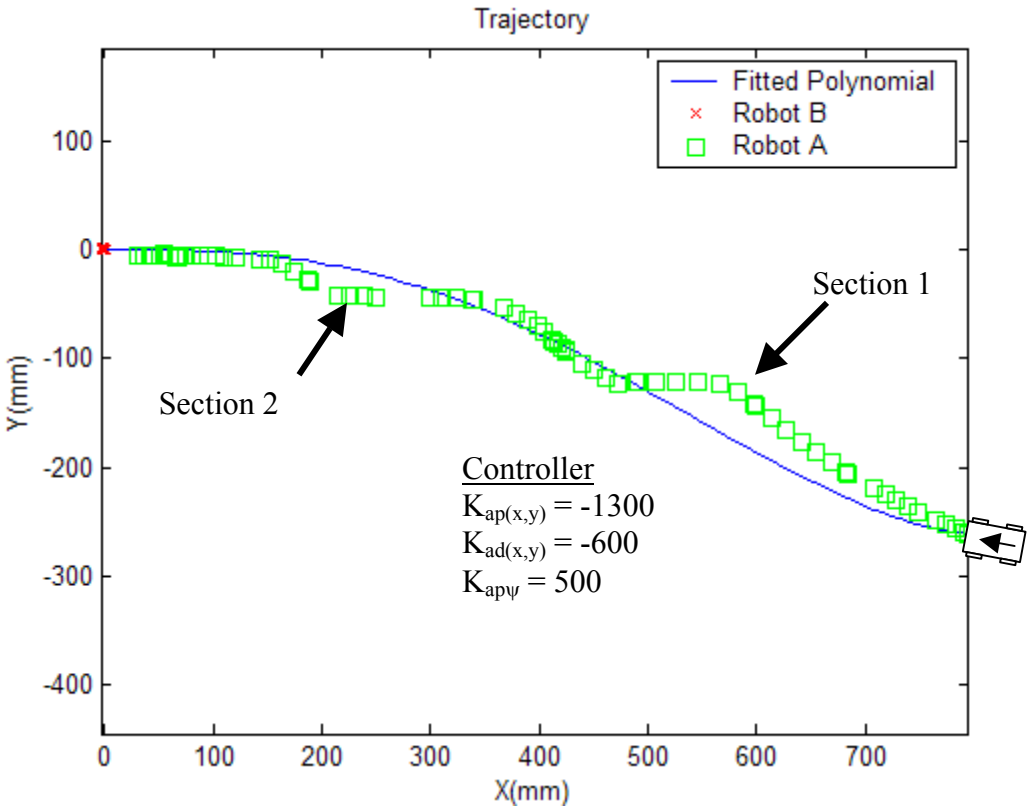

Figure 5.13. Movement of Robot A with too much heading ($K_{ap(x,y)}$ = -1300 $K_{ad(x,y)}$ = -600 $K_{ap\psi}$ = 500)

Figure 5.13 shows the results of one of the first trials. There is a noticeable deviation from the path that is caused by too much controller emphasis on the directional error. Section 1 shows where the robot was slightly behind in time and tried to follow the

slope instead of the position. When the error from the position became too large the position part of the controller took over and drove it back to the planned trajectory. Section 2 similarly illustrates where the robot wanted to straighten out its orientation for the final approach to its target. Again the position error became too large and pushed the vehicle back to its correct final position and orientation. Overall this run was not bad, but much improvement is needed.

When some of the coefficient for heading error is reduced figure 5.14 is produced. The movement of the robot is much smoother and deviates much less from the preplanned trajectory.

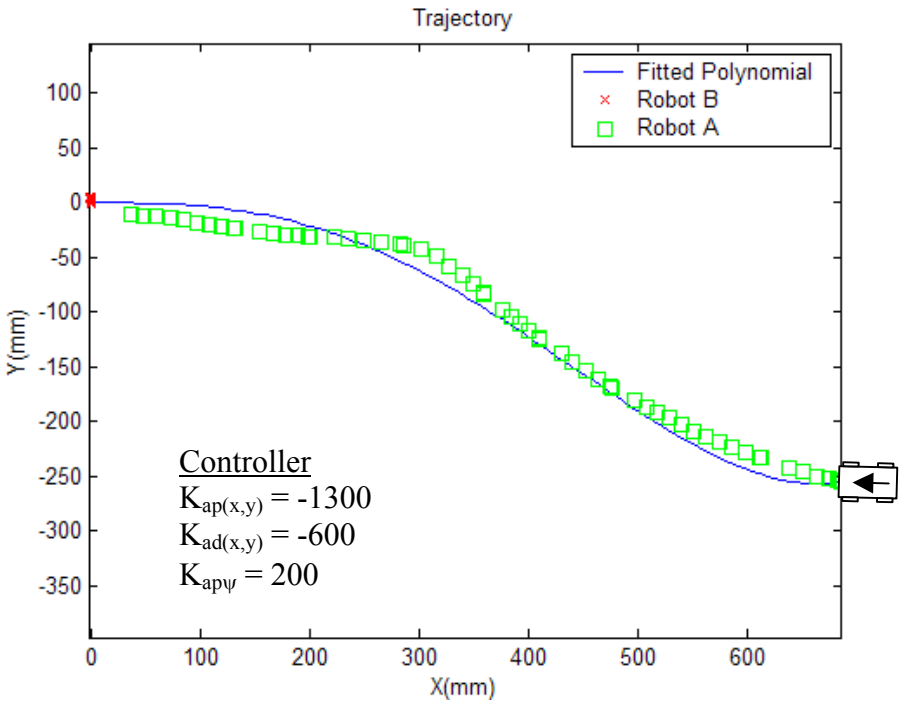

Figure 5.14. The system is improved by placing less effort on heading than the previous figure ($K_{ap(x,y)}$=-1300 $K_{ad(x,y)}$=-600 $K_{ap\psi}$=200)

The best controller tested works as given in figure 5.14. While there are some deviations from the path, the final orientation of the actual trajectory is accurate. The chosen control constants are given in table 5.6.

Table 5.6. Steering controllers and performance for Robot A

$K_{ap(x,y)}$	-1300
$K_{ad(x,y)}$	-600
K_{apy}	200
Maximum deviation (mm)	12
Final Position Error (mm)	4
Final Heading Error (rad)	0.05

5.4.2 Position and Heading Control for Robot B

Similar to the results given for Robot A, the controller for Robot B is best when the heading is reduced and the position controller is essentially the same as derived in section 5.2.2. Figure 5.15 shows the path of one of the controllers that was implemented. The position of the robot is not accurate, but the heading of the robot is acceptable. As Robot B approaches its target, the heading of the robot is correct, but the position of it is nearly 5cm below its goal.

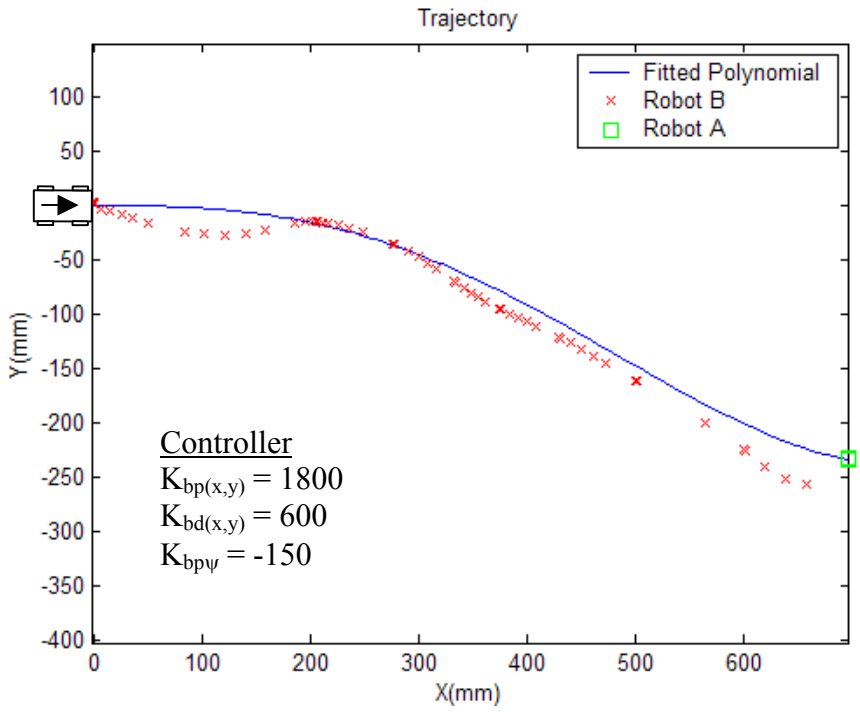

Figure 5.15. Robot B's controlled motion as it attempts to follow the given trajectory ($K_{bp(x,y)} = 1800$ $K_{bd(x,y)} = 600$ $K_{bp\psi} = -150$)

Similar to the procedure in selecting the superior controller for Robot A, the effects of the heading part of the controller for Robot B are reduced by changing $K_{bp\psi}$ from –150 to -10. Figure 5.16 shows the trajectory of the final implemented controller for Robot B. It is easily seen that the robot's motion are very close to the intended trajectory. The final choices for a good controller for Robot B are given in table 5.7. These values allow the robot to approach its target from several different poses with high accuracy. These various locations will be briefly seen in the following section as the two robots approach each other from opposite ends of the curve.

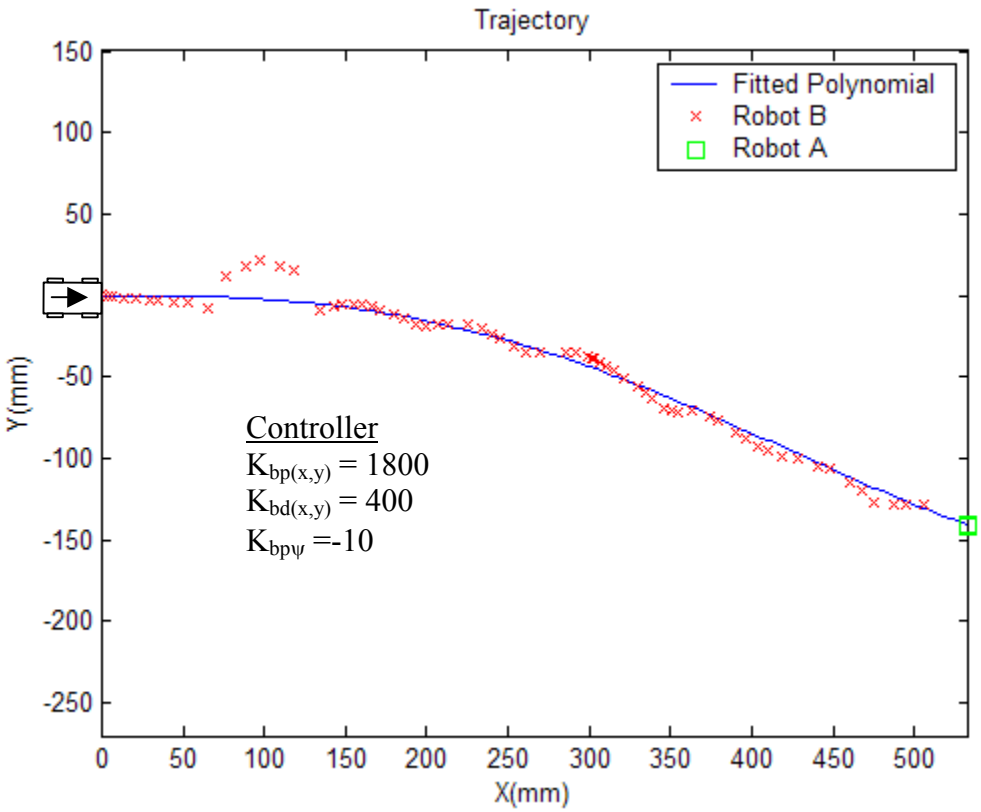

Figure 5.16. Robot B's controlled motion as it attempts to follow the given trajectory ($K_{bp(x,y)}$ = 1800 $K_{bd(x,y)}$ = 400 $K_{bp\psi}$ = -10)

Table 5.7. Steering controllers and performance for Robot B

$K_{bp(x,y)}$	1800
$K_{bd(x,y)}$	400
K_{bpy}	-10
Maximum deviation (mm)	25
Final Position Error (mm)	5
Final Heading Error (rad)	0.03

5.5 Coordination of Both Robots

The final discussion of the thesis will include a detailed analysis of the maneuvers of both robots as they traverse across the predetermined trajectory implementing all three controllers developed in the previous sections. Robot B will begin its trajectory from the left as Robot A correspondingly approaches from the right. The aspects of their coordination are developed when a decision is required to terminate their motion. The experiments ended when the robots were less than the predetermined 40mm apart (20mm for each robot).

The investigation begins with the acceptable controllers, though they were not chosen as the best design. Their performance will then be compared to the controller designated as superior. Figure 5.17 shows a good controller that allows the robots to dock together rather well. While Robot A does not follow the path well, its final pose with respect to Robot B is acceptable.

Figure 5.18 illustrates the difference when the control constants for heading are changed. The positions are relatively accurate with some deviations (caused by oscillations) by Robot B. The robots do however dock well. Their final position lies on the trajectory while their heading is slightly skewed.

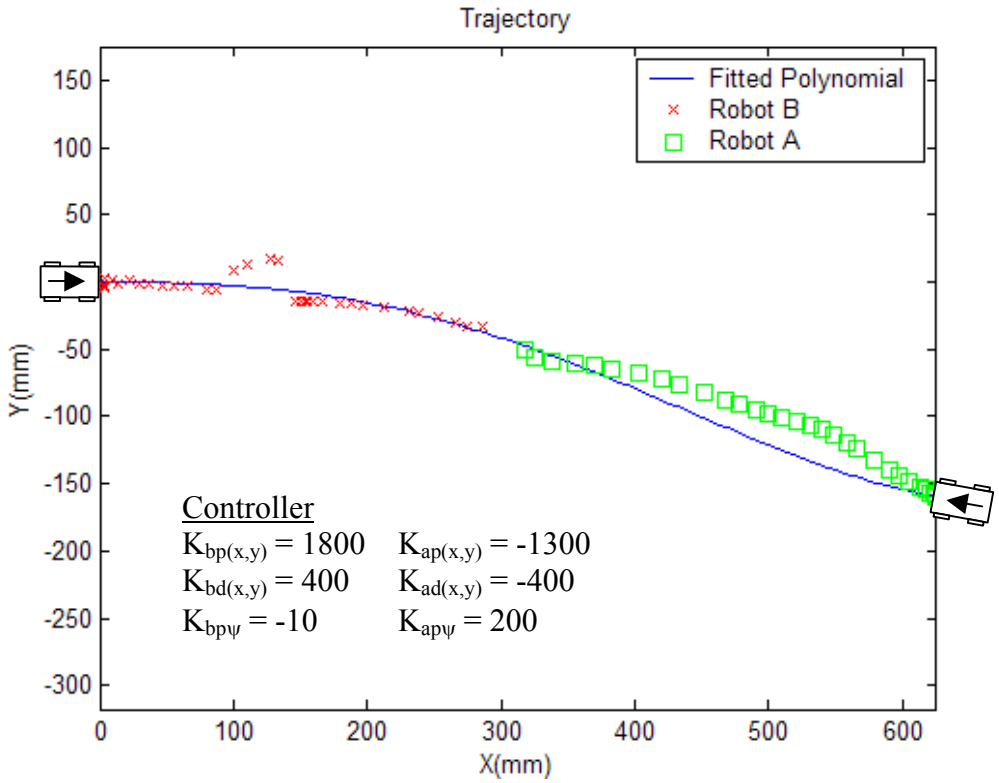

Figure 5.17. Tracking control of Robots A and B

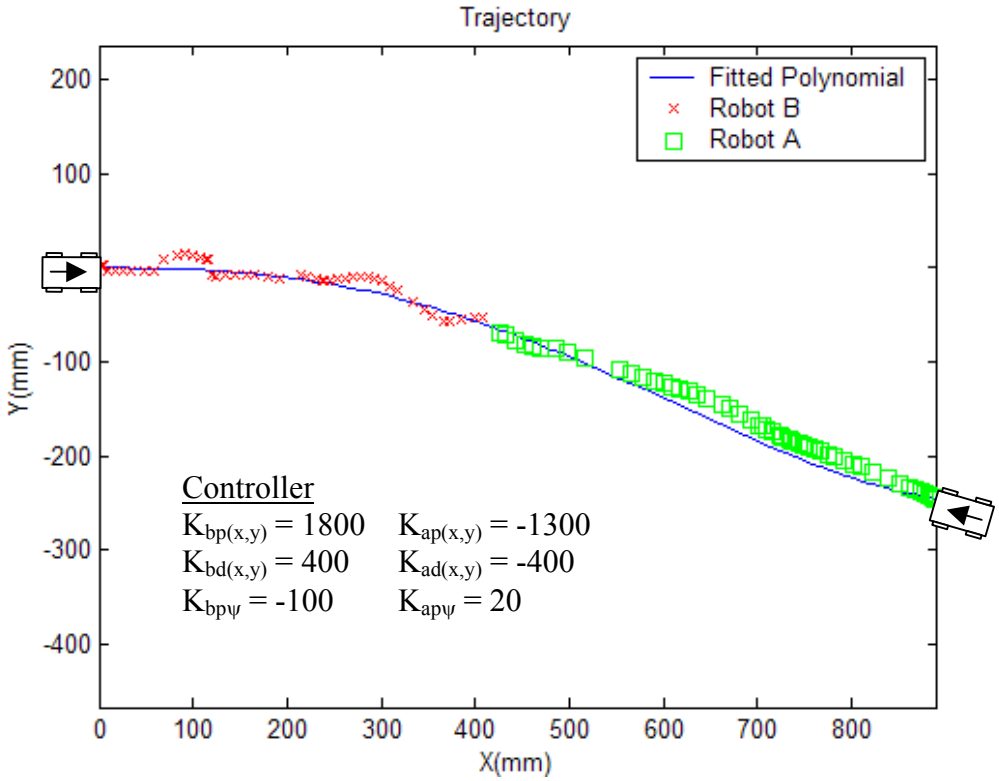

Figure 5.18. Tracking of Robots A and B

The controllers designed in section 5.4 proved to be the most accurate for the docking algorithm. Figure 5.19 shows the movements of the robots as Points A and B are tracked. Their motions along the path are highly accurate. Both their position and orientation follow the trajectory with minimal deviations. The bends in the curve remain as areas of difficulty. In the test run for figure 5.19 the largest deviation is 15 mm. These deviations become highly exaggerated along the turns in the polynomial.

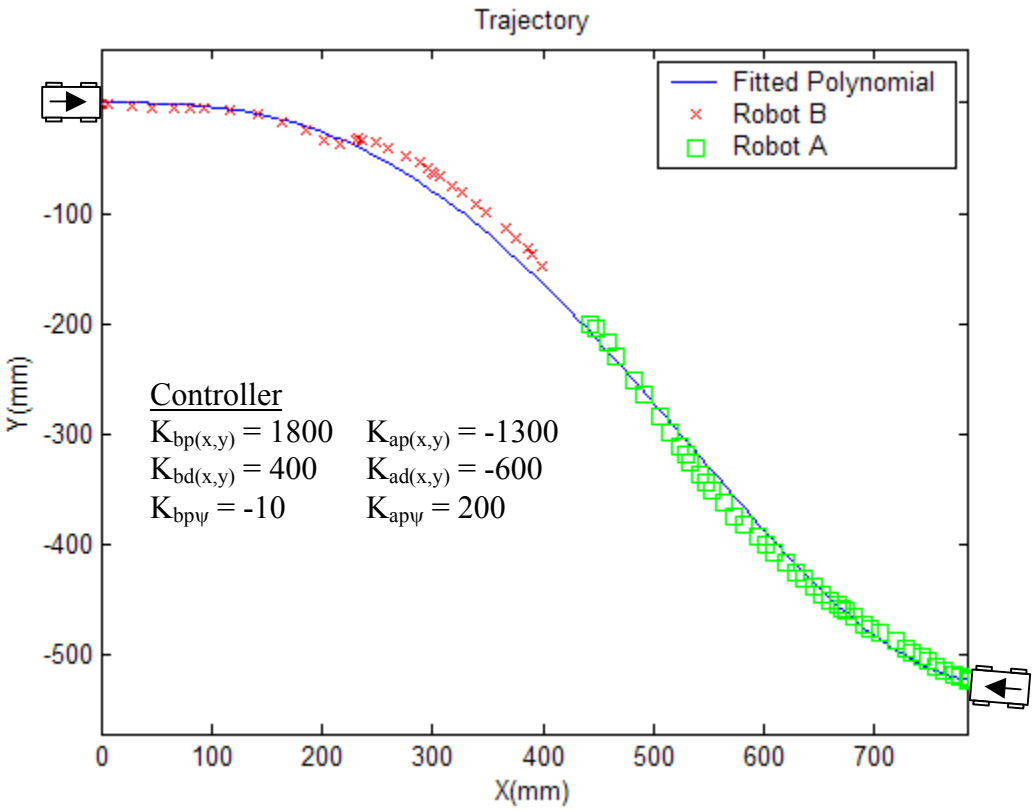

Figure 5.19. Coordination of both robots

Further tests were done to ensure that the chosen controller would repeatedly give good results for various tests. The top graphs (a and b) in figure 5.20 show the docking of the two robots when robot A begins in the 4th quadrant. Graphs (c and d) show docking from A's initial position in the 1st quadrant. All four sets show excellent tracking, which result in rather precise docking. This controller is therefore selected as

the superior controller by the robots' various maneuvers in rendezvous scenarios. The final control constants are given in table 5.8 with average performance results over 15 different tests from different poses. The final position error should be the same for both robots because their motion ends on the trajectory when they are within 40 mm of one another. However, this also considers deviations from the path, thus the errors are slightly different.

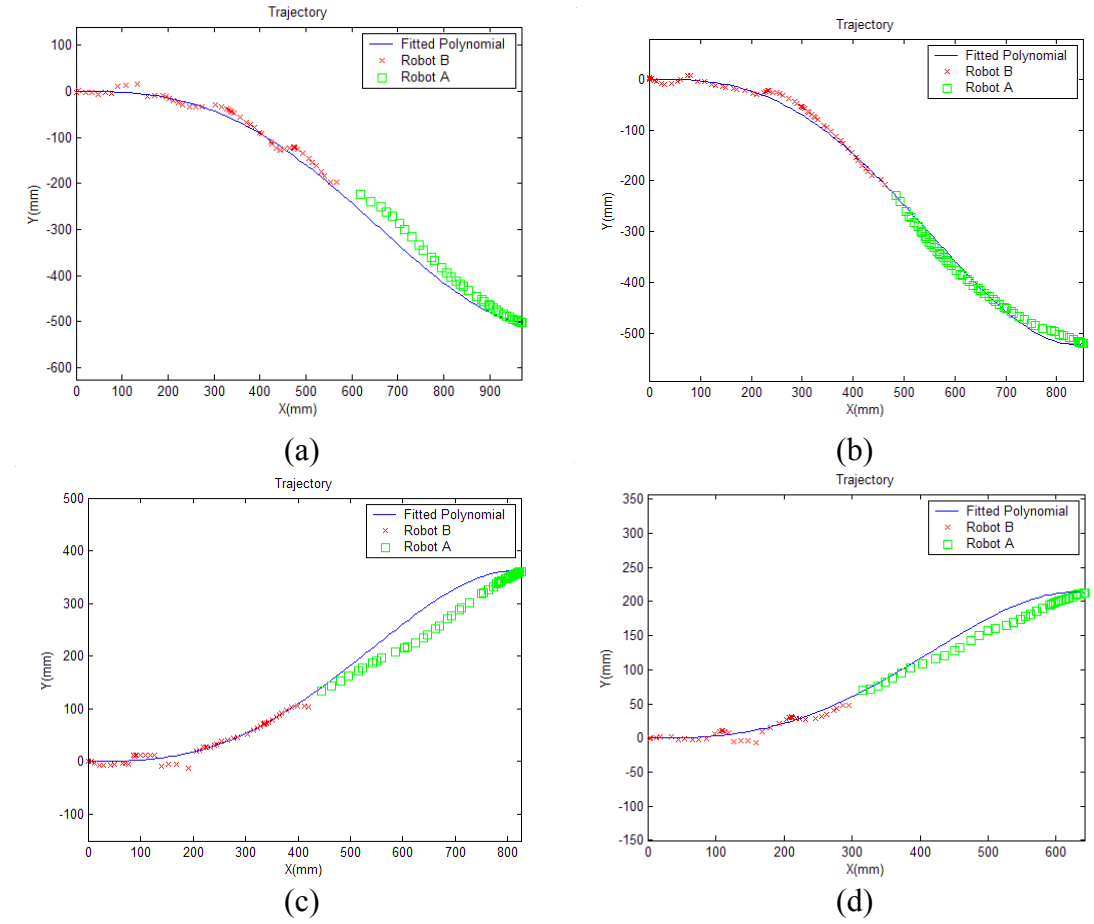

Figure 5.20. Multiple tests of the same controller to ensure control

Table 5.8. The chosen controller coefficients for docking

	Robot A	Robot B
$K_{p(x,y)}$	-1300	1800
$K_{d(x,y)}$	-600	400
$K_{p\psi}$	200	-10
K_{pspeed}	1100	-3000
K_{dspeed}	20	-30
Maximum deviation (mm)	21	12
Final Position Error (mm)	5.8	6.6
Final Heading Error (rad)	0.06	0.04

5.6 Limitation of Position Measurement System

It must be noted that while these results proved the effectiveness of the control algorithm, there were limitations on its capabilities because of frequent loss of LEDs. The figures given in the previous portions of this thesis were chosen specifically because of the high reliability of the position measurement system during those tests. This section focuses on the effects of not having proper position feedback (from lost LEDs). When the position of the robots cannot be determined (usually caused by loss of LEDs), the position of the system is not updated. The algorithm keeps running, but the position feedback data does not change until the LEDs are recovered. Figure 5.21 gives an excellent example of the loss of Robot B's position and the control recovery.

On the contrary, the system does not always have the capabilities to recover from such poor position feedback. Figure 5.22 shows an example when the position of Robot B was not rediscovered until corrections were not possible. The remainder of these examples is in appendix E.

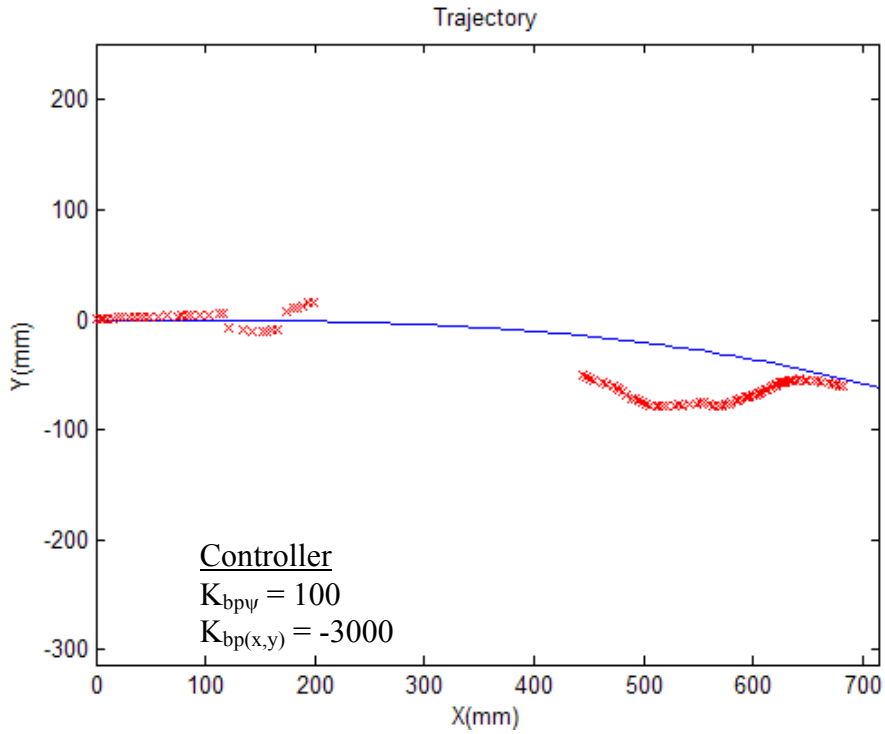

Figure 5.21. Regaining control after not detecting the position of Robot B

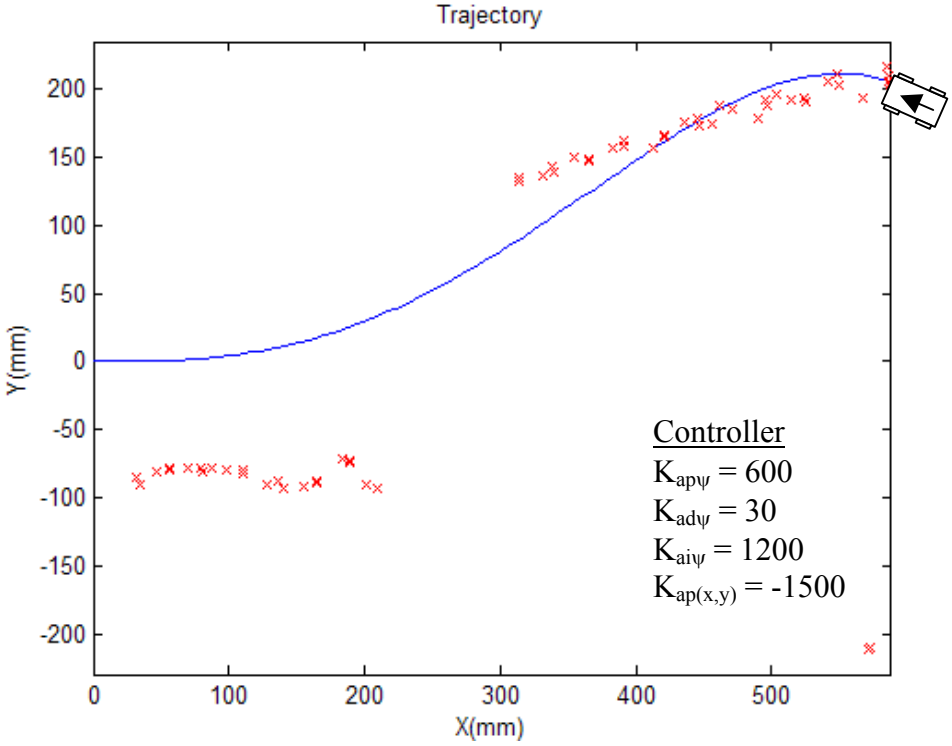

Figure 5.22. The loss of LEDs causes poor tracking for Robot A

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

This thesis has examined the application of a 4^{th} order polynomial for the planning of collaborative multi-agent robots. The implemented PD drive and steering controllers guarantee accurate path following per an acceptable 4^{th} order polynomial. The incorporation of both heading and position control assured that the robots' final pose would include both accurate position and heading.

While many goals such as multi-robot path planning for docking were accomplished in conducting the research of this thesis, there are many more problematic issues and future work that grew from this work. A review of the problem statement established in chapter 1 is performed.

This thesis will develop an autonomous path/ trajectory planning team of two multi-tasking robots in a controlled environment that share position data en route to docking with each other or individually to a stationary object while constrained by physical space, turning radii, and power limitations.

By incorporating the constraints of the nonholonomic robots a 4^{th} order polynomial was fit to determine a path that allowed the robots to dock with each by sharing position information. The goal of this thesis was therefore met. The 4^{th} order polynomial was created by determining the coefficients from the end conditions and minimum turning radii limitation. The bisection method was used to search over α_2 for maximizing the minimum radius of curvature, which converged quickly. Future study should include an analysis on the bisection method's convergence rate with comparison to other methods.

A better method may even include a symbolic solution to the five constraints of which minimum turning radius is the most cumbersome.

Another extremely important area that was not covered and should be addressed in the future includes looping or piecewise-defined polynomials. This would allow the robot that does not meet the criteria for the polynomial discussed in this work to make a loop or move in reverse to achieve a proper pose such that this work's 4^{th} order polynomial can be used.

Future work should also include sharing of knowledge about the workspace such as obstacle information. Docking in a workspace with sparse obstacles has numerous applications as discussed in chapter 1.

The camera system (pseudo GPS) used was extremely useful in the application of the presented tasks. Unfortunately, there were several problems such as frequently losing LEDs and a limited workspace. It would be extremely useful to have a larger physical workspace. Moving the cameras further apart or incorporating more cameras would enhance coverage.

Using trial-and-error for a highly non-linear system with no model, a good PID controller was determined. The presented controller illustrates that less sophisticated methods can be consistently applied. While the selected controller sufficiently verifies the algorithm implemented in this thesis, a model of the robots is certainly the next step in creating a more reliable, highly sophisticated controller after implementing more accurate gearboxes and steering mechanisms. Tuning methods were used as opposed to a model-based approach because the inaccuracies of the steering mechanism and drive gearbox would have negated the advantages achieved by the dynamical model obtained.

The results are promising but could be improved if other control methodologies were used (e.g. fuzzy control).

APPENDIX A
MATLAB CODE FOR CONTROLLING THE ROBOTS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Path Planning
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author: Chad Sylvester                                    %
% Date: October 23, 2003                                    %
%                                                           %
% This program allows for changing the center angles, control constants,       %
% and the minimum radius of curvature.  This program further sends commands    %
% out to the robots via the serial communication.  Feedback is received by     %
% reading two files that contain the position and heading of the robots        %
% (awhole.txt and bwhole.txt).  Several plots are also available for comparison %
% An option is also available to control a robot as remote control             %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc
% Center Steering Anlges
centerb=1375;
centera =1350;

%%%%%%%%%%% Control Constants %%%%%%%%%%%%%%%%%

% Heading Control for Robot B
Kbppsi = -100;
Tbdpsi = 0;
Tbipsi = 0;

% Position Control for Robot B
Kbppos = 1800;
Kbdpos = 400;
Kbipos = 0;

% Speed Control for Robot B
Kbpspeed = -3000;
Kbdspeed = -30;
Kbispeed = 0;

% Heading Control for Robot A
Kappsi = 20;
Tadpsi = 0;
Taipsi = 0;

% Position Control for Robot A
Kappos = -1300;
Kadpos = -400;
Kaipos = 0;

% Speed Control for Robot A
Kapspeed = 900;
Kadspeed = 0;
```

```
Kaispeed = 0;

% Initial Variable Settings
speed = 60;              % Set the average speed
rmin = 500;              % Set the minimum radius of curvature
epsilon = 10^-9;         % Set a small for the convergence of c
h=0;                   % Check for an available polynomial
maxminrad = rmin;        % Create a new variable for rmin that will be updated

% Initialize serial communications
% Construct the serial port object:
s1 = serial('COM1', 'BaudRate', 9600);

% To connect the serial port object to the serial port:
fopen(s1)

% Allows the user to decide whether to control the robots
% manually or use position feedback
for i = 1:10
   choice = input('(1) remote control  (2) feedback     ');
   if (choice == 1)|(choice == 2)
      break
   end
end

% Remote Controller for Robot A
% Depending on the users input the robot changes from steering or direction(F/R)
if choice == 1
   for i=1:100
      in = input('(1) forward (2) backward) (3) left) (4) right    ');
      if (in>0)&(in<5)
         if in == 1
            s = 30000;
         end
         if in == 2
            s = -30000;
         end
         if in == 3
            ang = 2000;
         end
         if in == 4
            ang = 1000;
         end
         % Send commands to robot via serial port
         angle = sprintf('%f',ang);
         speed = sprintf('%f',s);
         fprintf(s1, 'y');
         fprintf(s1, angle);
         fprintf(s1, 'o');
         fprintf(s1, speed);
         fprintf(s1, 'j');
      else
         % Reset steering to center, turn off motor, and quit program
         ang = centera;
         s = 0;
         angle = sprintf('%f',ang);
         speed = sprintf('%f',s);
         fprintf(s1, 'y');
         fprintf(s1, angle);
         fprintf(s1, 'o');
         fprintf(s1, speed);
         fprintf(s1, 'j');
```

```
        break
      end
    end
end

% Robots Controlled by Position Feedback
if choice == 2
   % Get pose of Robot A from file
   [A,B,C]=textread('awhole.txt','%f%f%f');
   xf = A(1);
   yf = A(2);
   ycheck = yf;
   psi = A(3);
   yp=tan(psi-pi);
   R = sqrt(xf^2+yf^2);

   % Calculate a path
   tic,  % timer that ends with toc
   dx=xf/1000;  % Set the differential element of x
   x1=0:dx:xf;
   cA=-.2;    % Initial guess for the left bound on c
   cB=.1;     % Initial guess for the right bound on c
   cAold=cA;   % Saves the previous value of the left bound
   q=0;
   for(i=1:1000)
      cAA(i)=cA;
      cBB(i)=cB;

      bA = (4*yf-yp*xf-2*cA*xf^2)/xf^3;
      aA = ((cA*xf^2+xf*yp-3*yf)/(xf^4));
      rA(i)=min(abs(((1+(4*aA.*x1.^3+3*bA.*x1.^2+2*cA.*x1).^2).^1.5)./(12*aA.*x1.^2+6*bA.*x1+2*cA)));

      bB = (4*yf-yp*xf-2*cB*xf^2)/xf^3;
      aB = ((cB*xf^2+xf*yp-3*yf)/(xf^4));
      rB(i)=min(abs(((1+(4*aB.*x1.^3+3*bB.*x1.^2+2*cB.*x1).^2).^1.5)./(12*aB.*x1.^2+6*bB.*x1+2*cB)));


      cA1=cA+epsilon;
      cB1=cB-epsilon;
      bA1 = (4*yf-yp*xf-2*cA1*xf^2)/xf^3;
      aA1 = ((cA1*xf^2+xf*yp-3*yf)/(xf^4));
      rA1=min(abs(((1+(4*aA1.*x1.^3+3*bA1.*x1.^2+2*cA1.*x1).^2).^1.5)./(12*aA1.*x1.^2+6*bA1.*x1+2*cA1)));

      bB1 = (4*yf-yp*xf-2*cB1*xf^2)/xf^3;
      aB1 = ((cB1*xf^2+xf*yp-3*yf)/(xf^4));
      rB1=min(abs(((1+(4*aB1.*x1.^3+3*bB1.*x1.^2+2*cB1.*x1).^2).^1.5)./(12*aB1.*x1.^2+6*bB1.*x1+2*cB1)));

      slopea=(rA1-rA(i))/epsilon
      slopeb=(rB(i)-rB1)/epsilon

      if and(slopea>0,slopeb<0)
         %disp('slope different')
         cAold=cA;
         cA=(cB-cA)/2+cA
         cB=cB
      else
         % disp('same slope rA > rB')
         cB=cA
         cA=cAold
      end

      if rA(i)>rmin;
```

```
        cuse=cA;
        ause=aA;
        buse=bA;
        h=1;
        rmin=rA(i);
      end
      if ((cB-cA)<epsilon)
        break;
      end
    end

    toc % End timer


    if (h==0)
      fclose(s1);
      'Polynomial is not available.'
      break;
    else
      rho=abs(((1+(4*ause.*x1.^3+3*buse.*x1.^2+2*cuse.*x1).^2).^1.5)./(12*ause.*x1.^2+6*buse.*x1+2*cuse));

      for j=1:1001
        check=0;
        y(j)=ause*x1(j)^4+buse*x1(j)^3+cuse*x1(j)^2;
        if (j>1)
          scurve(j)=scurve(j-1)+sqrt((y(j)-y(j-1))^2+(x1(j)-x1(j-1))^2);
        else
          scurve(1) = 0;
        end
      end
%       % Display the length of the curve
%       disp('Length of curve = ');
%       disp(scurve);
%
%       % Display the minimum radius of curvature for the polynomial
%       % that is defined by the selected c
%       disp('maxminrad = ');
%       disp(maxminrad);
    end

    % Find a time vector, t, given the length and desired average speed of the vehicle
    for i=1:length(x1)
      y1(i)=ause*x1(i)^4+buse*x1(i)^3+cuse*x1(i)^2;
      yp1(i)=4*ause*x1(i)^3+3*buse*x1(i)^2+2*cuse*x1(i);
      t(i)=x1(i)*(sqrt(1+(yp1(i))^2))/speed;
    end

    % Rewrite X(t) by doing a polynomial of t(x)
    t1 = 0:.25:t(length(x1));
    P = polyfit(t,x1,4);
    x2 = polyval(P,t1);
    Pyp = polyfit(t,yp1,4);
    yp2 = polyval(Pyp,t1);

    % Reverse order for Robot A (right to left)
    for i=1:length(x2)
      yp3(length(yp2)-i+1)=yp2(i);
      x3(length(x2)-i+1)=x2(i);
    end

    % Fit a polynomial for commanded position of Robot A
    P1 = polyfit(t1,x3,4);
```

```
% Plot desired trajectory.  The position data will be plotted on top of this graph
figure(1)
plot(x1,y1)
title('Trajectory')
xlabel('X(mm)')
ylabel('Y(mm)')
axis equal
hold

temp1 = clock;

% Wait for the users go ahead to continue
input('Press Enter');

% Set a bunch of parameters to 0 before tracking the robots
t0 = clock;
n=1;
Eapsi = 0;
iaEpsi = 0;
Eapos = 0;
iaEpos = 0;
Ebpsi = 0;
ibEpsi = 0;
Ebpos = 0;
ibEpos = 0;
R1=0;
Rb(1)=0;
Ra(1)=0;
Xa(1)=xf;
Xb(1)=0;
iR=0;
ta(1)=0;
tb(1)=0;
t2=0;
t3=0;
t(1)=0;
xaf(1)=xf;
xA=xf;
yaf(1)=yf;
xbf(1) = 0;
xB = 0;
ybf(1) = 0;
angb(1)=centerb;
anga(1)=centera;
dt = 0;

% Start a new timer
tic,
while(1)
   n=n+1;
   %%%%%%%%%%%%% Robot b %%%%%%%%%%%%%%%%%%%%%%%%%%
   % Get pose from file
   [A,B,C]=textread('bwhole.txt','%f%f%f');
   xbf(n) = A(1);
   xB=A(1);
   ybf(n) = A(2);
   if ((xA-xB)<20)
      for(i=1:10)
         fprintf(s1, 'n');
         fprintf(s1, '1350');
         fprintf(s1, 'a');
```

```
            fprintf(s1, '0.00');
            fprintf(s1, 'm');
        end
        disp('Xb = 0')
    else

        bpsi = A(3);
        ybp(n)=tan(bpsi);
        % Set commanded Position and Heading
        Xb(n) = polyval(P,t3);
        Yb(n) = ause*Xb(n)^4+buse*Xb(n)^3+cuse*Xb(n)^2;
        Ypb(n) = 4*ause*Xb(n)^3+3*buse*Xb(n)^2+2*cuse*Xb(n);

        % Find the time that has elapsed
        temptime = clock;
        tb(n) = etime(temptime,t0);
        dt = tb(n)-tb(n-1);
        if (tb(n)<max(t1))
            t3=tb(n)+dt;
            if t3>max(t1)
                t3=max(t1);
            end
        else
            t3=max(t1);
        end
        plot(xbf(n),ybf(n),'xr')
        R1=Rb(n-1);
        Rb(n) = sqrt((Xb(n)-xbf(n))^2+(Yb(n)-ybf(n))^2);
        Rb1=sqrt((xaf(n-1)-xbf(n))^2+(yaf(n-1)-ybf(n))^2);
        if Rb1<Rb(n)
            Rb(n)=Rb1;
        end
        Ebpsi1 = Ypb(n)-ybp(n);
        dbEpsi = (Ebpsi1-Ebpsi)/dt;
        ibEpsi = ibEpsi+dt*(Ebpsi1-Ebpsi);
        beta = atan((Yb(n)-ybf(n))/(Xb(n)-xbf(n)));
        Ebpos1 = beta-bpsi;
        dbEpos = (Ebpos1-Ebpos)/dt;
        ibEpos = ibEpos+dt*(Ebpos1-Ebpos);

        % Left is 1800 and Right is 1200
        if ((xA-xB)<200)
            Rb2(n) = sqrt((xaf(n-1)-xbf(n))^2+(yaf(n-1)-ybf(n))^2);
            speedb = Kbpspeed*Rb2(n)*sign(xaf(n)-xbf(n))+(Rb2(n)-R1)*Kbdspeed+iR*Kbispeed;
            R1 = Rb2(n);
        else
            speedb = Kbpspeed*Rb(n)*sign(xbf(n)-Xb(n))+(Rb(n)-R1)*Kbdspeed+iR*Kbispeed;
        end

        % Only update the steering if the robot has a positive speed
        if (speedb > 1000)
            % Set right and left limits
            minang = 1100;
            maxang = 1700;
            angb(n) = centerb +Kbppos*Ebpos1+Kbdpos*dbEpos+Kbipos*ibEpos+
Kbppsi*(Ebpsi1+Tbdpsi*dbEpsi+Tbipsi*ibEpsi);
            if (angb(n) > maxang)
                angb(n) = maxang;
            end
            if (angb(n) < minang)
                angb(n) = minang;
            end
```

```
    else
        angb(n)=angb(n-1);
    end
    iR = iR+Rb(n)*sign(Xb(n)-xbf(n));
    Ebpsi = Ebpsi1;
    Ebpos = Ebpos1;

    % Set max and min speeds
    maxspeed = 9000;
    minspeed = 0; %Don't go backwards

    if (speedb > maxspeed)
        speedb = maxspeed;
    end
    if (speedb < minspeed)
        speedb = minspeed;
    end

   % Send out commands to Robot B
    angle = sprintf('%f',angb(n));
    s = sprintf('%f',speedb);
    fprintf(s1, 'n');
    fprintf(s1, angle);
    fprintf(s1, 'a');
    fprintf(s1, s);
    fprintf(s1, 'm');
end
%%%%%%%%%%%%%%%%%%% Robot A %%%%%%%%%%%%%%%%%%%%55
% Get pose from file
[A,B,C]=textread('awhole.txt','%f%f%f');
xaf(n) = A(1);
xA=A(1);
yaf(n) = A(2);
if ((xA-xB)<10)
    Xa(n)=0;
    Ya(n)=0;
    ta(n)=etime(temptime,t0);
    anga(n)=anga(n-1);
    Ra(n) = sqrt((Xa(n)-xaf(n))^2+(Ya(n)-yaf(n))^2);
    for(i=1:10)
        fprintf(s1, 'y');
        fprintf(s1, '1350');
        fprintf(s1, 'o');
        fprintf(s1, '0');
        fprintf(s1, 'j');
    end
    disp('Xa = 0')
    break;
end

if (A(3)>0)
    apsi = (pi-A(3));
else
    apsi = -(pi+A(3));
end
yap(n)=tan(apsi);

% Set commanded Position and Heading
Xa(n) = polyval(P1,t2);
if (Xa(n)<0)
    Xa(n)=0; % Don't let the command for Robot A be negative
end
```

```
Ya(n) = ause*Xa(n)^4+buse*Xa(n)^3+cuse*Xa(n)^2;
Ypa(n) = 4*ause*Xa(n)^3+3*buse*Xa(n)^2+2*cuse*Xa(n);

% Find the time that has elapsed
temptime = clock;
ta(n) = etime(temptime,t0);
dt = ta(n)-ta(n-1);
if (ta(n)<max(t1))
   t2=ta(n)+dt;
   if t2>max(t1)
      t2=max(t1);
   end
else
   t2=max(t1);
   dt=0;
end

plot(xaf(n),yaf(n),'gs');
R1=Ra(n-1);
Ra(n) = sqrt((Xa(n)-xaf(n))^2+(Ya(n)-yaf(n))^2);
Ra1=sqrt((xaf(n)-xbf(n))^2+(yaf(n)-ybf(n))^2);
if Ra1<Ra(n)
   Ra(n)=Ra1;
end

Eapsi1 = Ypa(n)-yap(n);
daEpsi = (Eapsi1-Eapsi)/dt;
iaEpsi = iaEpsi+dt*(Eapsi1-Eapsi);

beta = atan(-(Ya(n)-yaf(n))/(Xa(n)-xaf(n)));

Eapos1 = beta-apsi;
daEpos = (Eapos1-Eapos)/dt;
iaEpos = iaEpos+dt*(Eapos1-Eapos);

% Left is 1800 and Right is 1200
if ((xA-xB)<200)
   Rb2(n) = sqrt((xaf(n-1)-xbf(n))^2+(yaf(n-1)-ybf(n))^2);
   speedb = Kbpspeed*Rb2(n)*sign(xaf(n)-xbf(n))+(Rb2(n)-R1)*Kbdspeed+iR*Kbispeed;
   R1 = Rb2(n);
else
   speeda = Kapspeed*Ra(n)*sign(xaf(n)-Xa(n))+(Ra(n)-R1)*Kadspeed+iR*Kaispeed;
end

% Only update the steering if the robot has a positive speed
if (speeda > 1000)
   % Set right and left limits
   minang = 1100;
   maxang = 1800;
   anga(n) = centera +Kappos*Eapos1+Kadpos*daEpos+Kaipos*iaEpos+
Kappsi*(Eapsi1+Tadpsi*daEpsi+Taipsi*iaEpsi);
   if (anga(n) > maxang)
      anga(n) = maxang;
   end
   if (anga(n) < minang)
      anga(n) = minang;
   end
else
   anga(n)=anga(n-1);
end
iR = iR+Ra(n)*sign(xaf(n)-Xa(n));
Eapsi = Eapsi1;
```

```
        Eapos = Eapos1;

        % Set max and min speeds
        maxspeed = 30000;
        minspeed = 0;  % Don't go backwards

        if (speeda > maxspeed)
           speeda = maxspeed;
        end
        if (speeda < minspeed)
           speeda = minspeed;
        end

        % Send out commands to Robot A
        anglea = sprintf('%f',anga(n));
        sa = sprintf('%f',speeda);
        fprintf(s1, 'y');
        fprintf(s1, anglea);
        fprintf(s1, 'o');
        fprintf(s1, sa);
        fprintf(s1, 'j');
    end    % end while
end

fprintf(s1, 'n');
fprintf(s1, '1350');
fprintf(s1, 'a');
fprintf(s1, '0');
fprintf(s1, 'm');
fprintf(s1, 'y');
fprintf(s1, '1350');
fprintf(s1, 'o');
fprintf(s1, '0');
fprintf(s1, 'j');

disp('A initial steering angle');
disp((anga(2)-1500)/40);
disp('B initial steering angle');
disp((angb(2)-1500)/40);

%Close COM1
fclose(s1);
toc

figure(1)
legend('Fitted Polynomial','Robot B','Robot A')

% lengths=max(scurve)
% Ramax=max(Ra)
% Rbmax=max(Rb)
%
% figure(2)
% plot(ta,anga)
% xlabel('Time');
% ylabel('Angle (degrees)');
% title('Steering Angle of Robot A');

% figure(3)
% plot(tb,angb)
% xlabel('Time');
% ylabel('Angle (degrees)');
% title('Steering Angle of Robot B');
```

```
% time=max(ta)
% figure(4)
% subplot(1,2,1)
% plot(ta,Xa,'b',ta,xaf,'g')
% xlabel('Time (seconds)');
% ylabel('X (mm)');
% title('X(t)');
% axis([0,max(ta),0,max(Xa)])
% legend('Commanded X position','Actual X position');
%
% subplot(1,2,2)
% plot(ta,Ra)
% xlabel('Time (seconds)');
% ylabel('R (mm)');
% axis([0,max(ta),Ra(1),max(Ra)])
% title('The deviation from commanded position');

% time=max(tb)
%
% figure(4)
% subplot(1,2,1)
% plot(tb,Xb,'b',tb,xbf(1:length(tb)),'g')
% xlabel('Time (seconds)');
% ylabel('X (mm)');
% title('X(t)');
% axis([0,max(tb),Xb(1),max(Xb)])
% legend('Commanded X position','Actual X position');
%
% subplot(1,2,2)
% plot(tb,Rb)
% xlabel('Time (seconds)');
% ylabel('R (mm)');
% axis([0,max(tb),Rb(1),max(Rb)])
% title('The deviation from commanded position');
```

APPENDIX B
VISUAL C++ PROGRAM FOR POSITION FEEDBACK

This appendix holds the code written in Visual C++ that takes the camera frame position of the LEDs and through coordinate transformations creates the E frame presented in chapter 2. The proprietary portion of the camera system software is not included. Phasespace did help with writing the first half of this code. After the LEDs were positions determined, the author wrote the rest of this code.

```
// This program tracks Points A and B and stores them in
// files to be accessed by the MATLab controller

#include <WINDOWS.H>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include "owl.h"

// Prototypes
double sign(double value);

// Globals
int counter = 0;
int ha = 0;
int hb = 0;
int iterations = 0;
int kf1 = 0;
int kl1 = 0;
double axcheck = 0;
double La = 0;
double Lb = 0;
double k = 16.0;
double kphi = 2.500;
double ktheta = 1.7500;
double kspeed = 15.00;
double desired_r = 327.00;
double desired_phi = 160.00;//198.00;
double desired_theta = 0.00;//20.00;
double rcheck = 0.00;
double rad = 500.00; //750.00
double ar[3] = {0.00,0.00,0.00};
```

```
double aru1 = 0.00;
double aru2 = 0.00;
double aru3 = 0.00;
double au1 = 0.00;
double au2 = 0.00;
double au3 = 0.00;
double atheta = 0.00;
double apsi = 0.00;
double bu[3] = {0.00,0.00,0.00};
double br1 = 0.00;
double br2 = 0.00;
double br3 = 0.00;
double bRmag = 0.00;
double btheta = 0.00;
double bpsi = 0.00;
double bru1 = 0.00;
double bru2 = 0.00;
double bru3 = 0.00;
double b1u1 = 0.00;
double b1u2 = 0.00;
double b1u3 = 0.00;
double b2u1 = 0.00;
double b2u2 = 0.00;
double b2u3 = 0.00;
double amag = 0.00;
double bmag = 0.00;

double pi = 3.1415926;
            double e1u1=0;
            double e1u2=0;
            double e1u3=0;
            double zero1=0;
            double zero2=0;
            double zero3=0;
            double e2u1=0;
            double e2u2=0;
            double e2u3=0;
            double e1u[3]= {0.00,0.00,0.00};
            double e2u[3]={0.00,0.00,0.00};
            double zero[3]= {0.00,0.00,0.00};

     double b2x = 0.00;
            double b2y = 0.00;
            double b2z = 0.00;
            double b2mag = 0.00;

// change these to match your configuration
#define BOX_NUMBER 0x20
#define MARKER_COUNT 10
#define SERVER_NAME "10.227.244.79"

// for hard real time
#define INIT_FLAGS 0

int n =0;
// for soft real time
```

```
//#define INIT_FLAGS OWL_POSTPROCESS

struct print_error {
  int err;
  print_error(int err) : err(err) { }
};

ostream &operator<<(ostream &out, print_error p);
ostream &operator<<(ostream &out, OWLMarker &m);


void create_point_tracker(int tracker, int box_id, int marker_count)
{
  cout << "point tracker " << tracker
       << ", box 0x" << hex << box_id << dec
       << ", markers " << marker_count << endl;

  // create tracker
  owlTrackeri(tracker, OWL_CREATE, OWL_POINT_TRACKER);

  // set markers
  for(int i = 0; i < marker_count; i++)
    owlMarkeri(MARKER(tracker, i), OWL_SET_LED, LED(box_id, i));

  // activate tracker
  owlTracker(tracker, OWL_ENABLE);
}  // end create_point_tracker


int main(void)
{
            // Record Data
            FILE *file1;
            file1 = fopen("example1.txt", "w");
            FILE *filedata;
            filedata = fopen("filedata.txt", "w");
            FILE *fileawhole;
            fileawhole = fopen("c:\\matlabR12\\work\\awhole.txt", "w");
            FILE *filebwhole;
            filebwhole = fopen("c:\\matlabR12\\work\\bwhole.txt", "w");
            FILE *fileafront;
            fileafront = fopen("c:\\matlabR12\\work\\afront.txt", "w");
            FILE *filebfront;
            filebfront = fopen("c:\\matlabR12\\work\\bfront.txt", "w");
            OWLMarker markers[32];

            if(owlInit(SERVER_NAME, INIT_FLAGS) < 0) return 0;

            // create tracker 0
            create_point_tracker(0, 0x20, 5);
            create_point_tracker(1, 0x21, 5);

            // check for errors
            if(!owlGetStatus()){
                        int err = owlGetError();
                        cerr << "error in point tracker setup: " << print_error(err) << endl;
```

```
        return 0;}

// set default frequency and start streaming
owlSetFloat(OWL_FREQUENCY, OWL_MAX_FREQUENCY);

// Set the matrix of markers equal to 0
float markermatrix[10][3]=  {{0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00},
                                                {0.00, 0.00, 0.00}};

// main loop
for(;;){
        // Set the matrix of markers equal to 0
        int w,v;
        for (v=1;v<11;v++){
                for(w=1;w<4;w++){
                        markermatrix[v][w]=  0.00;
                }
        }
        int err;

        // get some markers
        int n = owlGetMarkers(markers, 32);


        // check for error
        if((err = owlGetError()) != OWL_NO_ERROR){
                cerr << "error: " << print_error(err) << endl;
                break;}

        // no data yet.  wait for a bit
        if(n == 0){
                timeval t = {0, 1000};
                select(0, 0, 0, 0, &t);
                continue;}

        if(n > 0){
                //fprintf(file1,"10 markers\n");
                for(int i = 0; i < n; i++){
                   if(markers[i].cond > 0){
                                markermatrix[i+1][1] = markers[i].x;
                                markermatrix[i+1][2] = markers[i].y;
                                markermatrix[i+1][3] = markers[i].z;
                        } // end if
                        fprintf(file1, "%f   %f   %f", markers[i].x, markers[i].y, markers[i].z);
                  //cout << i << ") " << markers[i] << endl;

                }  // end for int
        }  // end if n>0
```

```
fprintf(file1,"\n\n");
// Calculations for unit vector of a
// Box 20 is numbers 1 - 5
// Box 21 is numbers 6 - 10
ha=0;
int kf1 = 1;
if(markermatrix[kf1][1]>=0.001 || markermatrix[kf1][1]<=-.001){
        ha=1;}
//for (int kf2 = kf1+1;kf2<6;kf2++){
int kf2 = 5;
if (ha==1){
        ha=0;
        if(markermatrix[5][1]>=0.001 || markermatrix[5][1]<=-.001){
          ha=1;}
}
if (ha==1){
        ha=0;
if (kf1==1){
        La = 30;}

double ax = markermatrix[5][1]-markermatrix[1][1];
double ay = markermatrix[5][2]-markermatrix[1][2];
double az = markermatrix[5][3]-markermatrix[1][3];
amag = sqrt(ax*ax+ay*ay+az*az);
au1 = ax/amag;
au2 = ay/amag;
au3 = az/amag;
if (abs(axcheck-ax)<50.0){
        ha=1;}
else{
        ha=0;
        ax=axcheck;
}
axcheck=ax;
if (iterations>15){
        fprintf(filedata,"%f %f %f\n",au1-zero1,au2-zero2,au3-zero3);
        //fprintf(fileafront,"%f  %f
%f",markermatrix[5][1],markermatrix[5][2],markermatrix[5][3]);

}
//fprintf(filedata,"a %f %f %f\n",au[1],au[2],au[3]);
//printf("au %f %f %f\n",ax,ay,az);
}
hb=1;
// Calculations for unit vector of b1
if (hb==1){
        hb=0;
        //for (kl1 = 6;kl1<8;kl1++){
        kl1 = 6;
        if(markermatrix[6][1]>=0.001 || markermatrix[6][1]<=-.001){
                hb=1;}
        //for (int kl2 = kl1+1;kl2<9;kl2++){
        int kl2 = 8;
        if (hb==1){
                hb=0;
                if(markermatrix[8][1]>=0.001 || markermatrix[8][1]<=-.001){
```

```
                                        hb = 1;}
                        }

                        Lb = 30;

                        double b1x = markermatrix[8][1]-markermatrix[6][1];
                        double b1y = markermatrix[8][2]-markermatrix[6][2];
                        double b1z = markermatrix[8][3]-markermatrix[6][3];
                        double b1mag = sqrt(b1x*b1x+b1y*b1y+b1z*b1z);
                        b1u1 = b1x/b1mag;
                        b1u2 = b1y/b1mag;
                        b1u3 = b1z/b1mag;

                }


                // Calculations for unit vector of b2
                int kb1 = 0;
                int kb2 = 0;
                if (hb==1) {
                        hb=0;
                        if(markermatrix[10][1]>=0.001 || markermatrix[10][1]<=-.001){
                                if(markermatrix[6][1]>=0.001 || markermatrix[6][1]<=-.001){
                                        kb1 = 6;
                                        kb2 = 10;
                                        hb=1;}
                        }

                        b2x = markermatrix[kb2][1]-markermatrix[kb1][1];
                        b2y = markermatrix[kb2][2]-markermatrix[kb1][2];
                        b2z = markermatrix[kb2][3]-markermatrix[kb1][3];
                        b2mag = sqrt(b2x*b2x+b2y*b2y+b2z*b2z);
                        b2u1 = b2x/b2mag;
                        b2u2 = b2y/b2mag;
                        b2u3 = b2z/b2mag;
                        if (hb==1){
                                hb=0;
                                if (b2mag>0){
                                hb=1;}
                        }
                }

                if (iterations>15){
                        fprintf(filebfront,"%f  %f
%f",markermatrix[8][1],markermatrix[8][2],markermatrix[8][3]);}

                if ((hb==1)&(ha==1)){
                iterations = iterations + 1;
                if ((iterations <  15)&(iterations>3)){
                        e1u1=b1u1+e1u1;
                        e1u2=b1u2+e1u2;
                        e1u3=b1u3+e1u3;
                        printf("e1u = %f %f %f\n",e1u1,e1u2,e1u3);
                        zero1=markermatrix[8][1]+Lb*e1u1/(iterations-3);
                        zero2=markermatrix[8][2]+Lb*e1u2/(iterations-3);
                        zero3=markermatrix[8][3]+Lb*e1u3/(iterations-3);
```

```
                printf("zeros = %f %f %f\n",zero1,zero2,zero3);
                e2u1=b2u1+e2u1;
                e2u2=b2u2+e2u2;
                e2u3=b2u3+e2u3;
                printf("e2u = %f %f %f\n",e2u1,e2u2,e2u3);
        }
        if (iterations ==14){
                e1u1=e1u1/11;
                e1u2=e1u2/11;
                e1u3=e1u3/11;
                e2u1=e2u1/11;
                e2u2=e2u2/11;
                e2u3=e2u3/11;
                zero1 = zero1;
                zero2 = zero2;
                zero3 = zero3;
                printf("e1u = %f %f %f\n",e1u1,e1u2,e1u3);
                printf("e2u = %f %f %f\n",e2u1,e2u2,e2u3);
                printf("zero = %f %f %f\n",zero1,zero2,zero3);

        }}
        if (iterations>15){

                // Calculations for r for Robot a
                if (ha==1){
                        double arx = markermatrix[5][1]+La*au1-zero1;
                        double ary = markermatrix[5][2]+La*au2-zero2;
                        double arz = markermatrix[5][3]+La*au3-zero3;
                        double armag = sqrt(arx*arx+ary*ary+arz*arz);
                        //printf("ARmag = %f\n",armag);
                        ar[1] = arx;
                        ar[2] = ary;
                        ar[3] = arz;
                        aru1 = arx/armag;
                        aru2 = ary/armag;
                        aru3 = arz/armag;
                        double a5mag = sqrt((markermatrix[5][1]-zero1)*(markermatrix[5][1]-
zero1)+(markermatrix[5][2]-zero2)*(markermatrix[5][2]-zero2)+(markermatrix[5][3]-
zero3)*(markermatrix[5][3]-zero3));


                        // Calculations for theta of Robot a
                        atheta =
acos(aru1*e1u1+aru2*e1u2+aru3*e1u3)*sign(aru1*e2u1+aru2*e2u2+aru3*e2u3);

                        // Use the dot product to find the psi for Robot a
                        apsi =
acos(au1*e1u1+au2*e1u2+au3*e1u3)*sign(au1*e2u1+au2*e2u2+au3*e2u3);

                        double Xa;
                        Xa = (armag*cos(atheta));

                        double Ya;
                        Ya = (armag*sin(atheta));

                        // Save to file whole.txt
```

```
                                fseek(fileawhole,0L,SEEK_SET);
                                fprintf(fileawhole, "%f\n",Xa);
                                fprintf(fileawhole, "%f\n",Ya);
                                fprintf(fileawhole, "%f",apsi);
                                fseek(fileafront,0L,SEEK_SET);
                                fprintf(fileafront,"%f\n%f%",a5mag*cos(atheta),a5mag*sin(atheta));
                        }

                        // Calculations for r for Robot b
                        if (hb==1){
                                double brx = markermatrix[6][1]+Lb*b1u1-zero1;
                                double bry = markermatrix[6][2]+Lb*b1u2-zero2;
                                double brz = markermatrix[6][3]+Lb*b1u3-zero3;
                                double brmag = sqrt(brx*brx+bry*bry+brz*brz);
                                br1 = brx;
                                br2 = bry;
                                br3 = brz;
                                bru1 = brx/brmag;
                                bru2 = bry/brmag;
                                bru3 = brz/brmag;

                                // Calculations for theta for Robot b
                                btheta =
acos(bru1*e1u1+bru2*e1u2+bru3*e1u3)*sign(br1*e2u1+br2*e2u2+br3*e2u3);

                                // Use the dot product to find the psi for Robot b
                                bpsi =
acos(b1u1*e1u1+b1u2*e1u2+b1u3*e1u3)*sign(b1u1*e2u1+b1u2*e2u2+b1u3*e2u3);

                                double Xb;
                                Xb = (brmag*cos(btheta));

                                double Yb;
                                Yb = (brmag*sin(btheta));

                                // Save to file whole.txt
                                fseek(filebwhole,0L,SEEK_SET);
                                fprintf(filebwhole, "%f\n",Xb);
                                fprintf(filebwhole, "%f\n",Yb);
                                fprintf(filebwhole, "%f",bpsi);
                        }

                        if (kbhit()){
                                break;}
                }
        } //end forever
fclose(fileawhole);
fclose(filebwhole);
fclose(filedata);

// cleanup
owlDone();
return 0;
} //end main

// printing operators
```

```
ostream &operator<<(ostream &out, print_error p)
{
 if(p.err > 0)
   switch(p.err) {
   case OWL_NO_ERROR: out << "No Error"; break;
   case OWL_INVALID_VALUE: out << "Invalid Value"; break;
   case OWL_INVALID_ENUM: out << "Invalid Enum"; break;
   case OWL_INVALID_OPERATION: out << "Invalid Operation"; break;
   default: out << "0x" << hex << p.err << dec; break;
   }
 else out << p.err;

 return out;
}

ostream &operator<<(ostream &out, OWLMarker &m)
{
 out << m.x << " " << m.y << " " << m.z;
 return out;
}

double sign(double value){
        double sign = 1;
        if (value<0){
                sign = -1;}
        return sign;
}
```

APPENDIX C
ROBOT SOFTWARE

```
// dock.c
// This program takes instruction from the controller
// Author: Chad Sylvester
// Date: October 23,2003

//include files
#include "global.h"
#include <avr/io.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include "serial.h"
#include "motor1.h"


// function prototypes
void wait(int waittime);
void io_init(void);
void setPortBit(char port, int pin);
void clearPortBit(char port, int pin);

// Globals
int minspeed = -20000;
int maxspeed = 20000;
int minang = 1000;
int maxang = 2000;

void setPortBit(char port, int pin){
            if (port=='B') {
              PORTB |= (1<<pin);
            }
}

void clearPortBit(char port, int pin){
            if (port=='B')       {
              PORTB &= ~(1<<pin);}
}


void io_init(void)
 {
       DDRB = 0xff;// Most of PORTB is used for motors and servos
       DDRE = 0xff; // more servos
                        //PORTE = 128;
 }

void wait(int waittime){ // Create a waiting function
```

```
        int time1, time2, time3;
        for (time1 = 0; time1 < waittime; time1++) {
        for (time2 = 0; time2 < 500; time2++){
        for (time3 = 0; time3 < 50; time3++);
        }}
}

void main(void){
wait(500);
io_init();
pwm_init();
int ang=1550;
servo(1,ang);
int speed = 0;
motor(2,speed);

USART0_init(103);
wait(50);

int delay=100;
unsigned int port = 0;
int n = 0;
for(;;){

get_numbers();
for(n=1;n<4;n++){
speed=speed+(SPEED-speed)*.50;
motor(2,speed);
wait(7);}


for(n=1;n<4;n++){
ang=ang+(ANGLE-ang)*.50;
servo(1,ang);
wait(7);}

}
}

#include <serial.c>
#include <motor1.c>
```

```
// motor1.h
// This is the header file for motor1.c
// Author: Chad Sylvester
// October 23, 2003

void pwm_init(void);
void servo(int servonumber,int angle);
void motor(int motornumber,int speed);


// motor1.c
// This the program that sends commands to the servo and DC motor
// Author: Chad Sylvester
// Date: October 23, 2003

#include "motor1.h"

 void pwm_init(void) {

        TCCR1A = 170;
        TCCR1B = 0x12;

        ICR1 = 20000;

        TCCR3A = 170;
        TCCR3B = 0x12;

        ICR3 = 20000;
                        // Set the Steering angle at 1500
 }

void servo(int servonumber,int angle) {
                        if (angle>maxang)
                        {
                        angle = maxang;
                        }

                        if (angle<minang)
                        {
                        angle = minang;
                        }
                        if (servonumber == 1){
        OCR1A = angle;}
                        if (servonumber == 2){
        OCR1B = angle;}
                        if (servonumber == 3){
        OCR1C = angle;}
                        if (servonumber == 4){
        OCR3A = angle;}
                        if (servonumber == 5){
        OCR3B = angle;}
                        if (servonumber == 6){
        OCR3C = angle;}
        }

void motor(int motornumber,int speed) {
```

```
if (motornumber == 2){
if (speed>0){
cbi(PORTE,7);
}

if (speed<0){
sbi(PORTE,7);
speed=-1*speed;
}

if (speed>maxspeed){
        speed = maxspeed;}

if (speed<minspeed){
        speed = minspeed;}

OCR1B = speed;
}


if (motornumber == 1){
if (speed>0){
cbi(PORTB,0);
}

if (speed<0){
sbi(PORTB,0);
speed=-1*speed;
}
if (speed>maxspeed){
        speed = maxspeed;}

if (speed<minspeed){
        speed = minspeed;}

OCR1A = speed;

}
}
```

```
// serial.h
// Header file for serial.c
// Author: Chad Sylvester
// Date: October 23,2003

//function prototypes

void USART0_init(uint8_t baudrate);
unsigned int USART0_receive(void);
void USART0_receive_char(void);
void USART0_transmit(uint8_t data);
void get_numbers(void);

void USART1_init(uint8_t baudrate);
unsigned int USART1_receive(void);
void USART1_transmit(uint8_t data);


// serial.c
// This the program that sends and receives information from the serial ports
// Author: Chad Sylvester with help from Max Billingsley
// Date: October 23, 2003

#include "serial.h"

void USART0_init(uint8_t baudrate){
// set buad rate

UBRR0H = (unsigned char) (baudrate >> 8);
UBRR0L = baudrate;

// enable TX and RX

UCSR0B |= (1 << RXEN0) | (1 << TXEN0);

// 8 data bits, 1 stop bit, no parity

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}

unsigned int USART0_receive(void){
while (!(UCSR0A & (1 << RXC0)));

return UDR0;
}


void USART0_receive_char(void){
int i = 0;
while (!(UCSR0A & (1 << RXC0)));
inp[i]=UDR0;
i=i+1;}


void USART0_transmit(uint8_t data){
while (!(UCSR0A & (1 << UDRE0)));
```

```
UDR0 = data;
}


void get_numbers(void){
double dec=0.0;
double sign=1.0;
double numbera = 00.0;
double rbyte;
for(;;){
rbyte = USART0_receive();
if (rbyte==97){
numbera=numbera*sign;
break;}

//USART0_transmit(rbyte);
if (rbyte==110){
numbera=0.0;
dec=0.0;
sign=1;
}

if (rbyte==45)
            {if(numbera==0)
            {
            sign=-1.0;}
            }

if(rbyte>=46){
  if(rbyte<=57){

            if (dec>=1){
            if(rbyte!=46){
            dec=dec*10;

            numbera=numbera+(rbyte-48)/dec;}}

            if(rbyte==46){
            dec=1.0;}

            if (dec==0){
            numbera=numbera*10.0+rbyte-48.0;          }
            }}
}

dec=0.0;
sign=1.0;
double numberr = 00.0;
rbyte=0;

for(;;){
rbyte = USART0_receive();
if (rbyte==109){
numberr=numberr*sign;
break;}
```

```
//USART0_transmit(rbyte);
if (rbyte==110){
numberr=0.0;
dec=0.0;
sign=1;
}

if (rbyte==45)
            {if(numberr==0)
            {
            sign=-1.0;}
            }

if(rbyte>=46){
  if(rbyte<=57){

            if (dec>=1){
            if(rbyte!=46){
            dec=dec*10;

            numberr=numberr+(rbyte-48)/dec;}}

            if(rbyte==46){
            dec=1.0;}

            if (dec==0){
            numberr=numberr*10.0+rbyte-48.0;            }
            }}
}


ANGLE=numbera;
SPEED = numberr;
}

void USART1_init(uint8_t baudrate){
// set buad rate

UBRR1H = (unsigned char) (baudrate >> 8);
UBRR1L = baudrate;

// enable TX and RX

UCSR1B |= (1 << RXEN1) | (1 << TXEN1);

// 8 data bits, 1 stop bit, no parity

UCSR1C |= (1 << UCSZ11) | (1 << UCSZ10);
}

unsigned int USART1_receive(void){
while (!(UCSR1A & (1 << RXC1)));

return UDR1;
}
```

```
void USART1_transmit(uint8_t data){
while (!(UCSR1A & (1 << UDRE1)));

UDR1 = data;
}
```

APPENDIX D
DESIGN AND LAYOUT OF CIRCUIT BOARD FOR ROBOTS

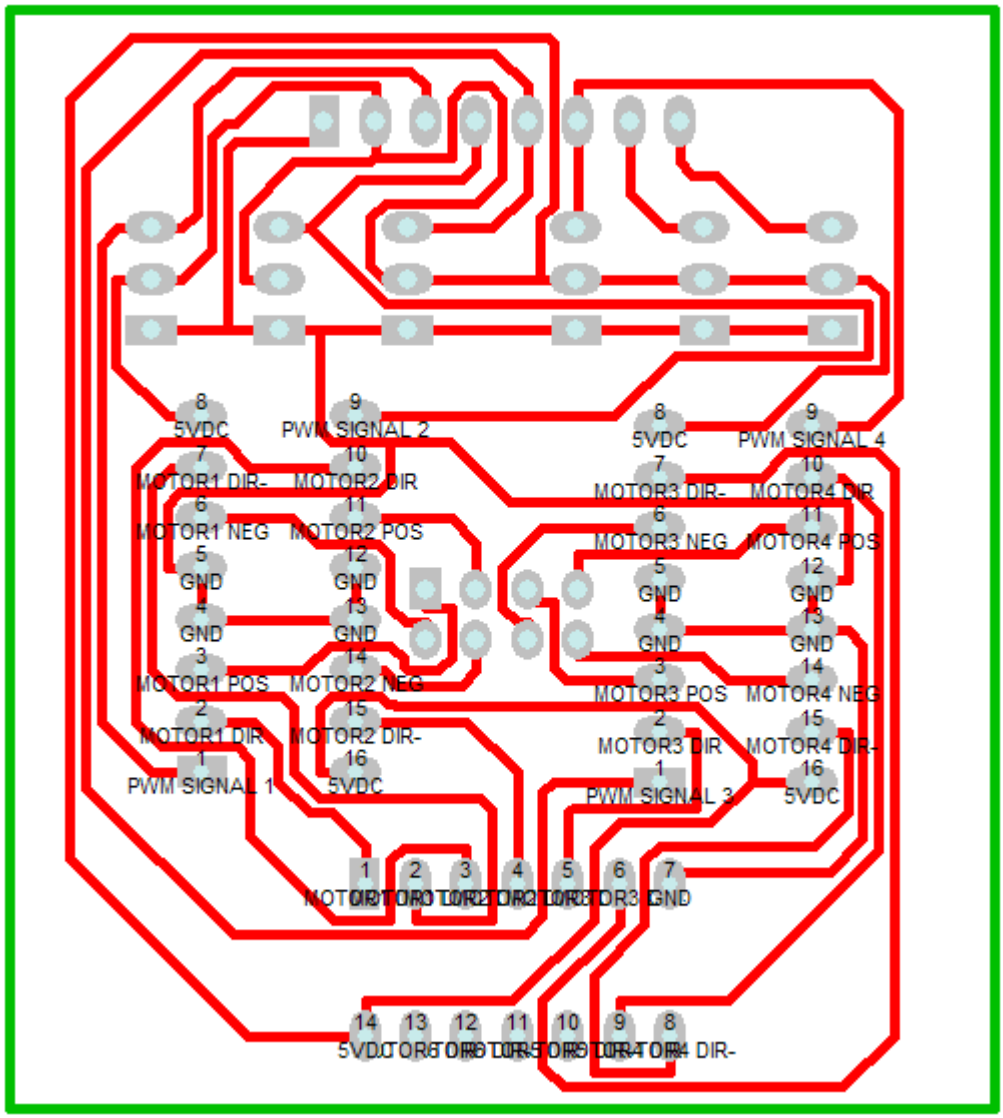

Figure D.1. Motor driver board connections

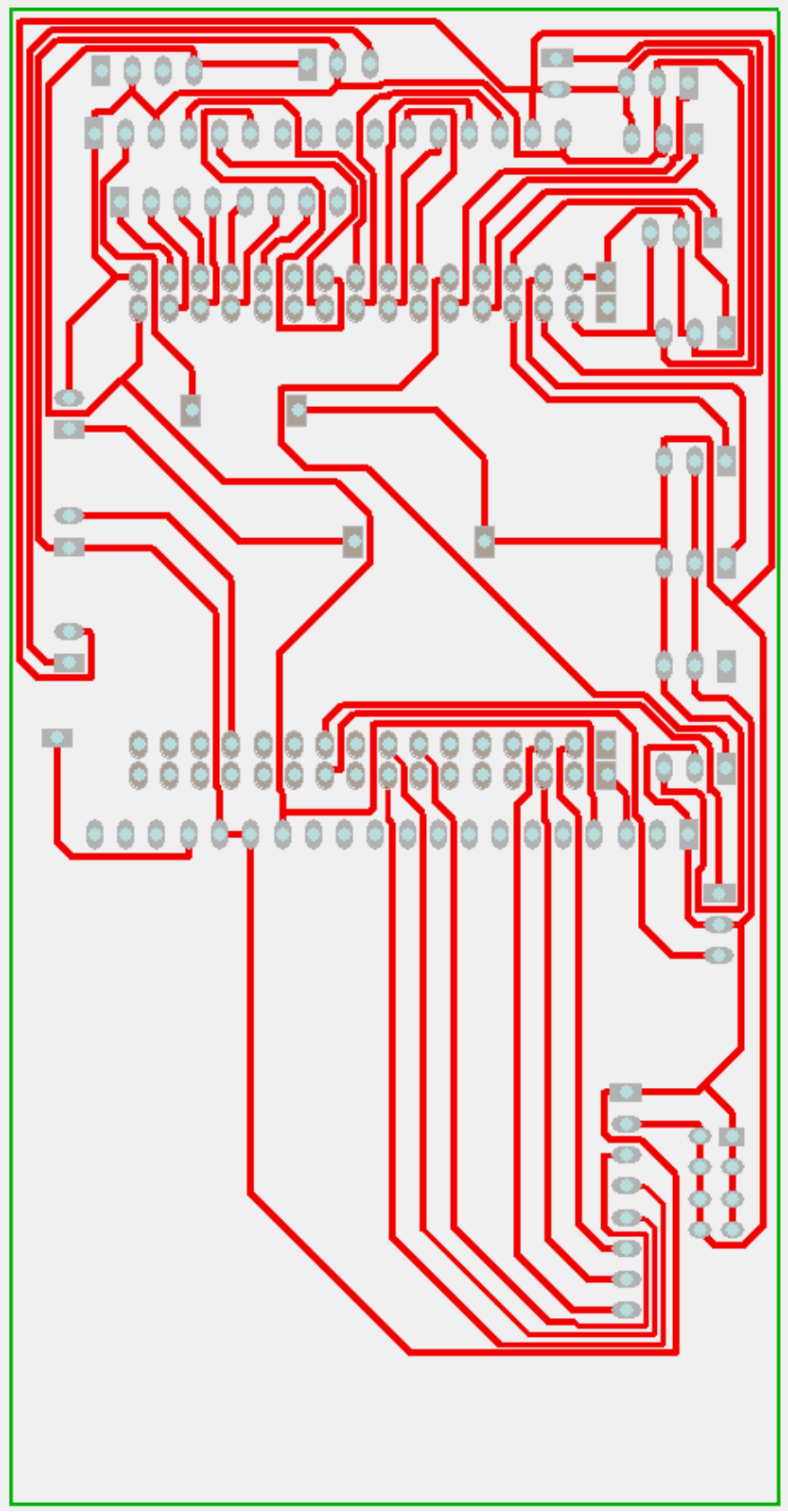

Figure D.2. Main board connections

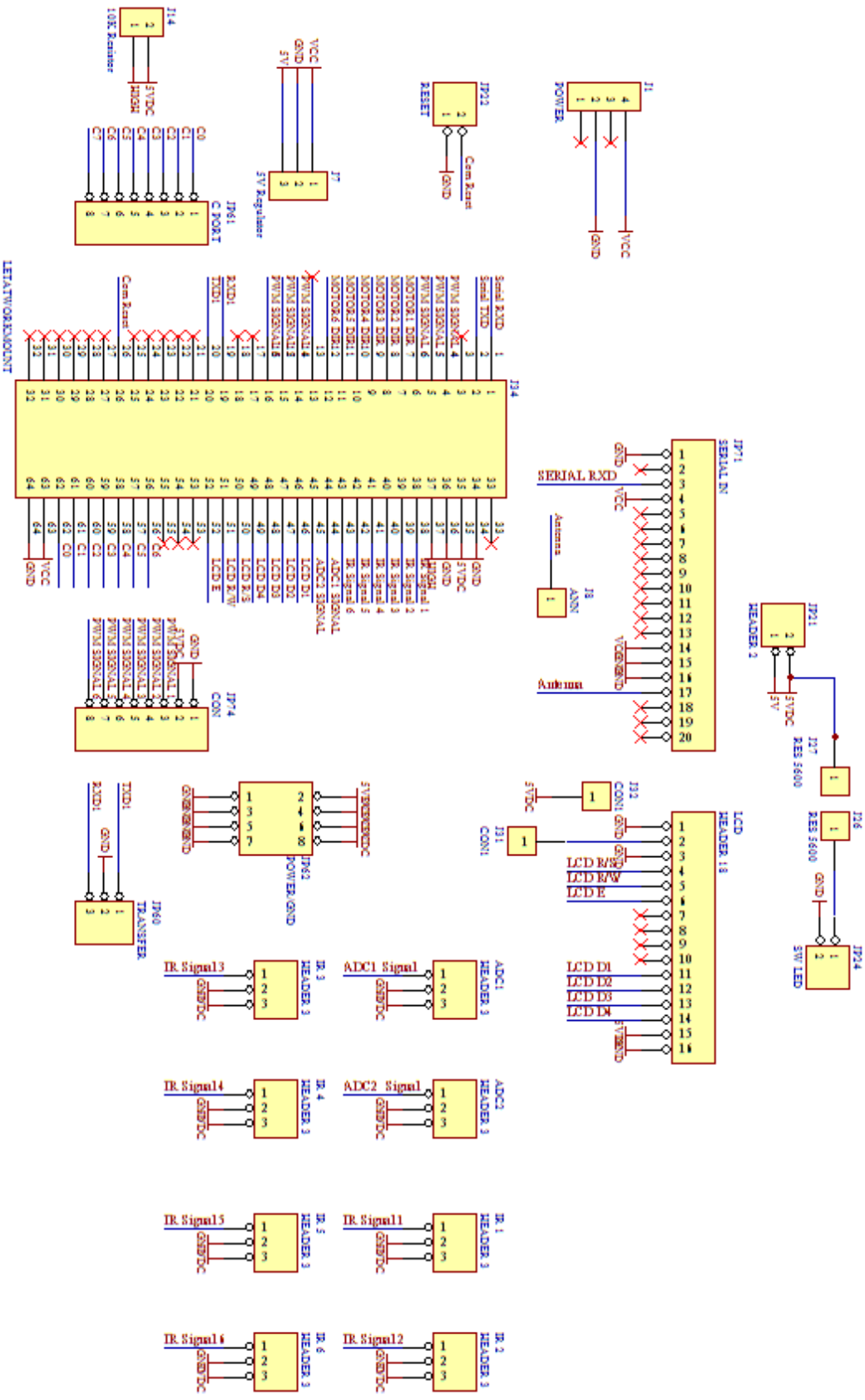

Figure D.3. Main board setup from Protel

APPENDIX E
RESULTS FROM LOST LED

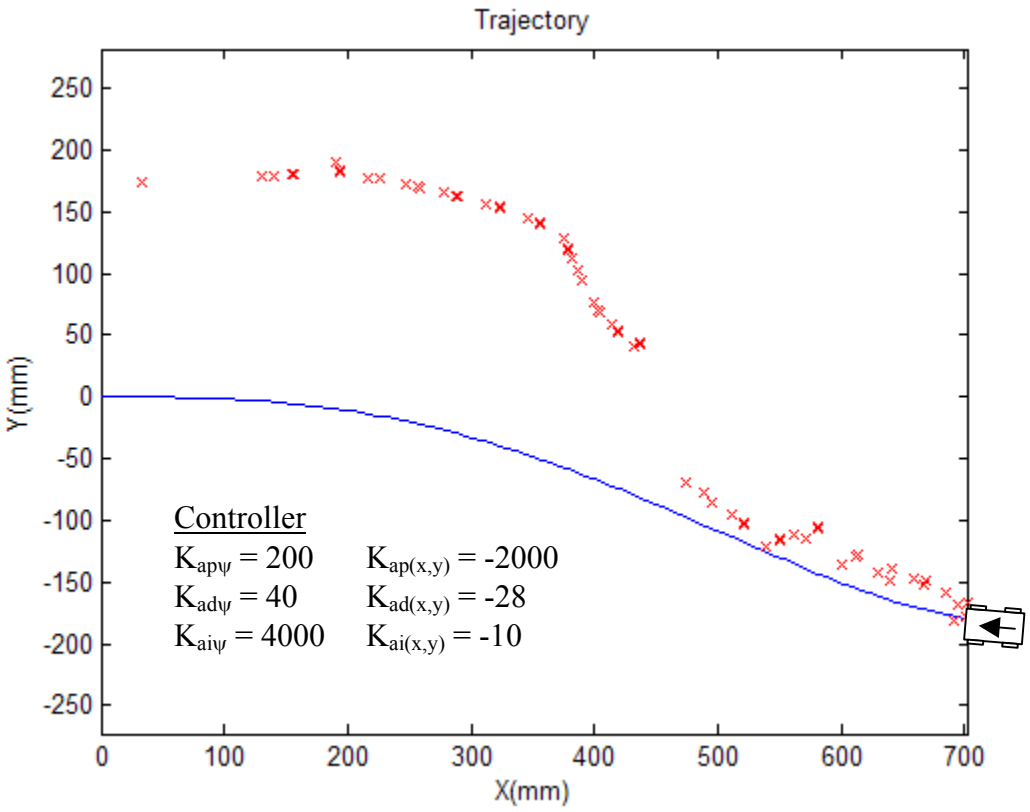

Figure E.1. Poor position feedback limits tracking ability

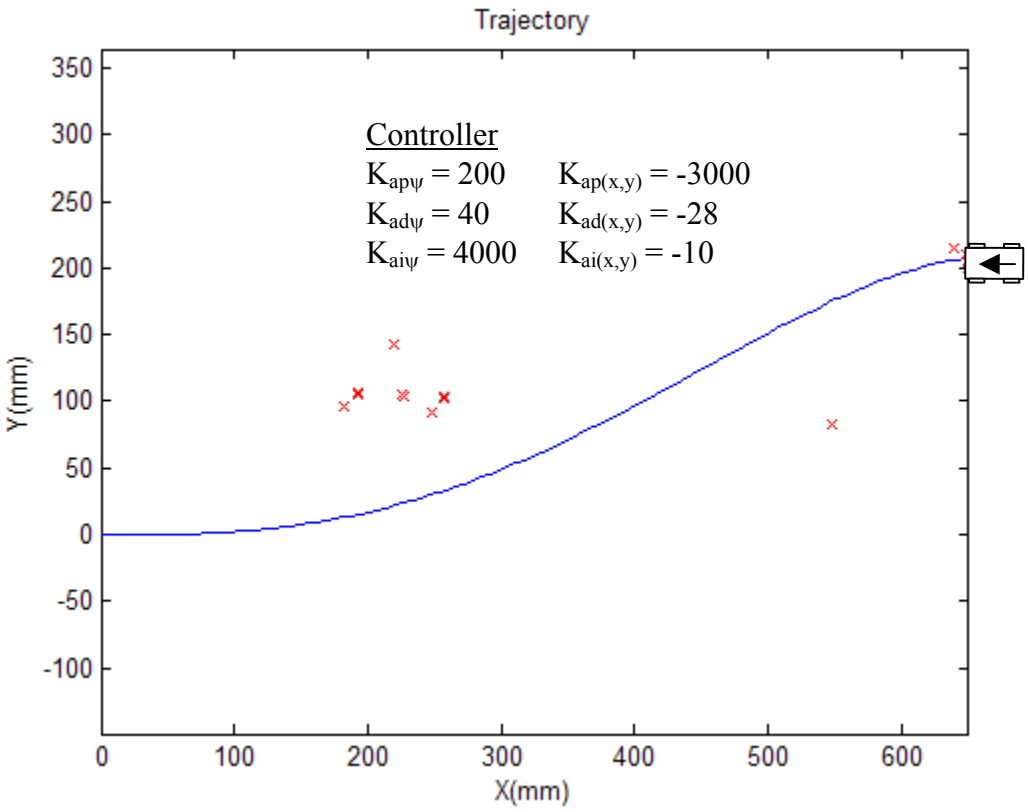

Figure E.2. Poor position feedback limits tracking ability. With only a few data points, tracking is impossible.

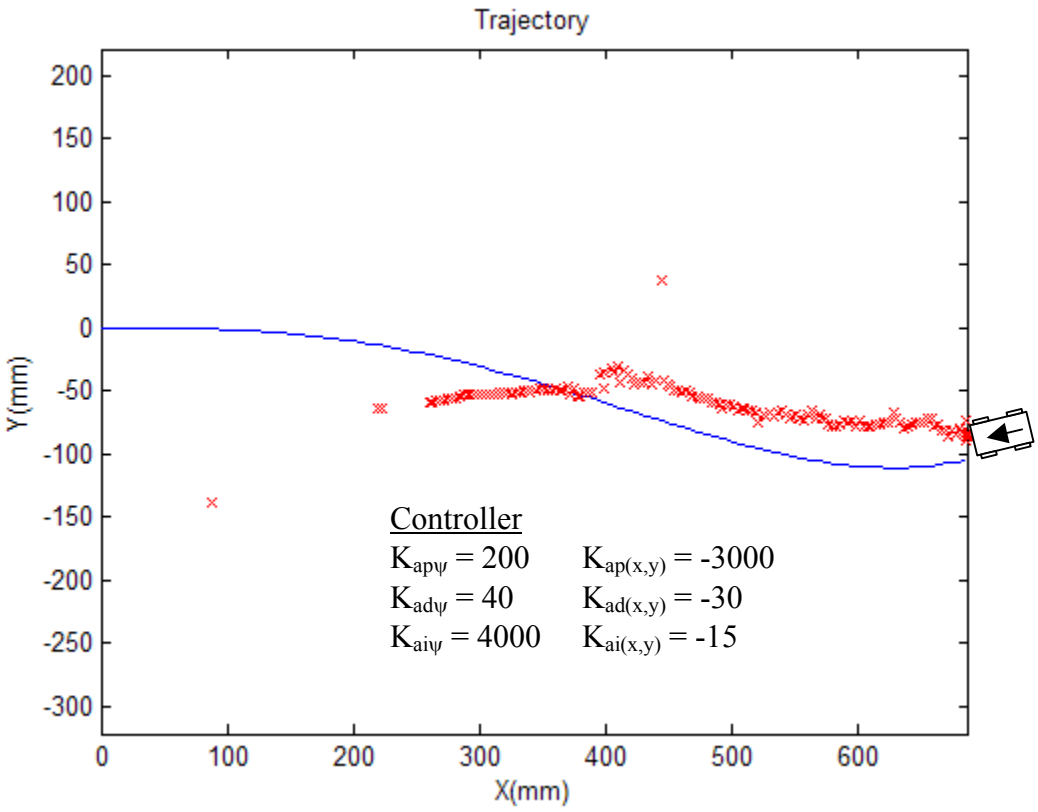

Figure E.3. Poor position feedback limits tracking ability. Losing the LEDs are detrimental to the tracking of the robots.

LIST OF REFERENCES

[Bru96] Brumitt, B.L. and Stentz, A.(April 1996). Dynamic Mission Planning for Multiple Mobile Robots, Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN.

[Bur00] Burgard, W., Moors, M., Fox, D., Simmons, R., and Thrun, S.(April 2000). Collaborative Multi-Robot Exploration, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA, pp 476-481.

[Car02] Carpin, S., Parker, L.(May 2002). Cooperative Leader Following in a Distributed Multi-Robot System, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 2994-3001.

[Cha97] Cha, Y. (1997). Navigation of a Free-Ranging Mobile Robot Using Heuristic Local Path-Planning Algorithm, Robotics and Computer-Integrated Manufacturing Vol. 13, No. 2, pp 145-156.

[Cha02] Chaimowicz, L., Campos, M., and Kumar, V.(May 2002). Dynamic Role Assignment for Cooperative Robots, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 293-298.

[Cho03] Choset, H. and Kortenkamp, D.Path Planning and Control for AERcam, a free-flying inspection robot in Space, http://www.ri.cmu.edu/projects/project_311.html visited on June 23, 2003.

[Das01] Das, A.K., Fierro, R., Kumar, V., Southall, B., Spletzer, J., and Taylor, C.J.(2001). Real-Time Vision-Based Control of a Nonholonomic Mobile Robot, International Conference on Robotics and Automation, Seoul, Korea, May 2001, pp. 1714-1719.

[Dix01] Dixon, W., Dawson, D., Zergeroglu, E. and Behal, A.(June 2001). Adaptive Tracking Control of a Wheeled Mobile Robot via an Uncalibrated Camera System, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics. Vol. 31, No. 3, pp 341-352.

[Don00] Donald, B., Gariepy, L., and Rus, D.(April 2000). Distributed Manipulation of Multiple Objects using Ropes, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA, pp 450-457.

[Ege01] Egerstedt, M. and Martin, C.(2001). Optimal Trajectory Planning and Smoothing Splines, Automatica, pp1057 – 1064 www.elsevier.com/locate/automatica visited on June 23, 2003.

[Eme02] Emery, R., Sikorski, K., Balch, T.(May 2002). Protocols for Collaboration, Coordination and Dynamic Assignment in a Robot Team, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 3008-3015.

[Gen00] Gentili, F. and Martineli, F.(April 2000). Robot group formations: a dynamic programming approach for a shortest path computation, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 3152-3157.

[Hwa02] Hwang, K. and Ju, M.(2002). Speed Planning for a Maneuvering Motion, Journal of Intelligent and Robotic Systems, Vol. 33 pp 25 - 44.

[Jag02] Jager, M. and Nebel, B.(May 2002). Dynamic Decentralized Area Partitioning for Cooperating Cleaning Robots, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 3577-3582.

[Kho02] Khoo, A. and Horswill, I.D.(May 2002). An Efficient Coordination for Autonomous Robot Teams, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 287-292.

[Kur00] Kurabayashi, D. and Asama, H.(April 2000). Knowledge Sharing and Cooperation of Autonomous Robots by Intelligent Data Carrier System, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 464-469.

[Lau97] Laugier, C., Garnier, P., Fraichard, T., Paromtchik, I. and Scheuer, A. Motion Planning and Sensor-Guided Maneuver Generation for an Autonomous Vehicle. Proceedings of the International Conference on Field and Service Robotics, pages 56-65, Canberra (AU), December 1997.

[Law00] Lawton, J., Young, B., and Beard, R.(April 2000). A Decentralized Approach to Elementary Formation Maneuvers, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 2728-2743.

[Lee02] Lee, J.H., Yi, B., and Suh, I.H.(May 2002). Impulse Measure Based Performance Analysis of Sawing Task by Dual Arm, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 982-988.

[Lei98] Lei, Z., Tasdizen, T., and Cooper, D.(January 1998). PIMs and Invariant Parts for Shape Recognition, 6th IEEE International Conference on Computer Vision, Bombay, India, pp 827 - 832.

[Miy00] Miyata, N., Ota, J., Aiyama, Y., Asama, H., and Arai, T.(April 2000). Cooperative Transport in Unknown Environment – Application of Real-time Task Assignment, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 3176-3182.

[Nag01] Nagy, B. and Kelly, A.(June 2001). Trajectory Generation for Car-Like Robots using Cubic Curvature Polynomials, Proceedings of the International Conference on Field and Service Robotics, Helsinki, Finland, pp 105-110.

[Ost02] Ostergaard, E., Mataric, M., and Sukhatme, G.(May 2002). Multi-robot Task Allocation in the Light of Uncertainty, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 3002-3007.

[Par96] Paromtchik, I., Laugier, C.(1996). Autonomous Parallel Parking of a Nonholonomic Vehicle, Proceedings of IEEE intelligent Vehicle Symposium, Tokyo, Japan, pp. 13-18.

[Par98] Paromtchik, I., Garnier, Ph., and Laugier, C.(1998). Motion Control for Autonomous Maneuvers of a Nonholonomic Vehicle, Intelligent Autonomous Systems. International Scientific Issue, Karlsruhe, Germany, pp. 38-45.

[Per02] Pereira, G., Pimentel, B., Chaimowicz, L., and Compos, M.(May 2002). Coordination of multiple mobile robots in an object carrying task using implicit communication, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 281-286.

[Pir00] Pirjanian, P., Mataric, M.(April 2000). Multi-Robot Target Acquisition using Multiple Objective Behavior Coordination, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 2696-2702.

[Sae01] Saedan, M. and Ang Jr., M.H., 3D Vision-Based Control of an Industrial Robot, Proceedings of the IASTED International Conference on Robotics and Applications, Nov 19-22, 2001, Florida, USA, pp 152-157.

[Yah98] Yahja, A., Stentz, A., Singh, S., and Brumitt, B.L. (May 1998). Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environment, Proceedings, IEEE Conference on Robotics & Automation, Leuven, Belgium.

[Yam00] Yamashita, A., Fuckuchi, M., Ota, J., Arai, T., Asama, H.(April 2000). Motion Planning for Cooperative Transportation of a Large Object by Multiple Robots in a 3D Environment, Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, pp 3144-3151.

[Zlo02] Zlot, R., Stenz, A., Dias, M., and Thayer, S.(May 2002). Multi-Robot Exploration Controlled by a Market Economy, Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington DC, pp 3016-3023.

BIOGRAPHICAL SKETCH

Chadwick Allen Sylvester was born April 13, 1980, in East Ridge, TN. He was reared in a godly, compassionate home by his loving parents. He was married on June 23, 2001, to his wife who devotedly assisted him in his endeavors to graduate with a Bachelor of Science degree in mechanical engineering in May 2002 from the University of Florida and his master's degree, which he also began in May.