uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Dave Gagné-Roussel

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Mechanical Engineering)

GRADE / DEGREE

Department of Mechanical Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Vision-based Navigation and Control of a Robotic Vehicle

TITRE DE LA THÈSE / TITLE OF THESIS

Dr. A. Fahim

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Dr. N. Baddour

Dr. R. Goubran

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Vision-Based Navigation and Control of a Robotic Vehicle

Dave Gagné-Roussel

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements of
MASTER OF APPLIED SCIENCES

in Mechanical Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Canada
June 2006

Your file Votre référence
ISBN: 978-0-494-25774-6
Our file Notre référence
ISBN: 978-0-494-25774-6

This thesis is dedicated to my twin brother, Dany, whose courage and determination are exemplary.

Abstract

A recurrent problem in mobile robotics is the difficulty to accurately estimate a robot's localization. The ability to successfully estimate the localization of a mobile robot is highly dependent on the type of sensory data used to infer its pose. Traditionally, this has been achieved with odometry and the integration of wheel encoders signals. A major drawback of this approach, however, is the inability to provide an accurate estimate of the heading orientation; a significant cause of odometry drift leading to navigation failure. Accordingly, there is a need for improved localization methods, and vision-based estimation holds promise for this purpose. This research proposes an alternative solution to pure odometry localization. To that end a visual pose estimation algorithm which combines robotic vision and odometry is proposed. The method utilizes scene image vanishing points for recovering the orientation of a mobile robot in a two-dimensional space. To assess the performance of the visual pose estimation algorithm on an operational prototype robotic vehicle developed in the course of the current research, an original pose tracking controller using the geometrical properties of Cardinal splines is implemented. The visual pose estimation algorithm is validated experimentally and compared against six sensory fusion schemes. The results show that the localization accuracy can be improved by one order of magnitude when compared to pure wheel encoder odometry. With regards to motion control, the pose tracking controller is also evaluated for the case of rectilinear trajectories. Future work on large-scale navigation strategies will be developed based on these ideas.

Acknowledgments

Over this two-year period, many people contributed to the successful completion of this research. Their help directly had an impact on my work. I would like to thank the most important ones.

First, I would like to express my sincere thanks to Dr. Atef Fahim for directing my studies. His extensive discussions and critics have had a positive influence on the final results of this research project. I am also in debt to the technical people of the Mechanical Department Workshop: Léo Denner, John Perrins, Brent Cotter, Michael Burns, and James MacDermid whose assistance was crucial during the prototyping phase of the robotic vehicle. Many thanks to my colleagues: Cagry, Ahmad, Ashan, and Ali whom have been of great company. Their presence during the harder days has been more than comforting. My special gratitude is also due to my family members for their unconditional love and support during this period.

Finally, I gratefully acknowledge the support of the Ontario Graduate Scholarship Program (OGS), the University of Ottawa, and all other contributors for partially funding this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The mere formulation of a problem is far more essential than its solution, which may be merely a matter of mathematical or experimental skills. To raise new questions, new possibilities, to regard old problems from a new angle requires creative imagination and marks real advances in science."

– Albert Einstein

Chapter 1

Introduction

1.1 Motivation

Mobile robotics has received much attention from scientists and academia during the past two decades. The increase of processing capability and the availability of low-cost electronic components contributed to the advancement of mobile robotics to a level of maturity allowing further development. This is especially true, as the foreseen demand for service robotics is increasing.

One of the major market drivers of service robotics is domestic core automation. As the *2005 World Robotics Survey* – prepared by the United Nations Economic Commission for Europe (UNECE) [1] – estimates the projected sales of domestic robots for the coming period of 2004-2007 to 4.5 million units. For the same period, the Survey also reports a similar trend in robotic solution for professional use. It estimates to 50 000 the number of new installations. Professional robotic applications with strong growth include underwater systems, medical systems, defense and rescue, as well as security applications. Mobile robotic platforms alone account for 12% or some 6 000 of the projected units. Figure 1.1 gives a comprehensive list of

professional service robots and the projected installations for the period 2004-2007.

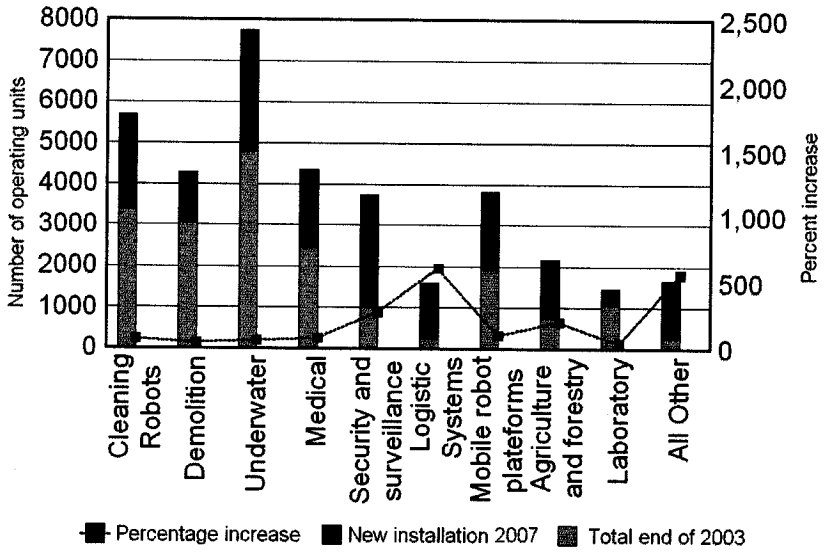


Figure 1.1: Service robots for professional use. Stock at the end of 2003 and projected installations in 2004-2007. Reproduced from [1].

Many reasons justify this sudden increase in the use of robotics. Among those is the quest for increase productivity, the transfer of dangerous and laborious tasks from man to machine, as well as improving quality of life [1]. The increase is also in part attributed to the international climate and the need for improving security of airports, public institutions, and private facilities such as factory and utility plants.

Although, active research in the field has been successfully conducted, commercial, mobile robotics is relatively new. Clearly, the development of autonomous solution is deemed to grow. As these figures demonstrate, the key to this endeavor is to reach the mass market and shift from high-cost prototypes to solutions consumers and industrials can afford. In that respect, a fair avenue for advances in robotic mobility is vision. Robotic vision has the

potential of changing the traditional odometry-based navigation approach and its associated drawbacks.

1.2 Problem Statement

Localization is a recursive problem in the field of mobile robotics. Numerous approaches have already been proposed in an attempt to solve this problem. The more common ones utilize sonar for detecting obstacles and motion sensors for odometry purposes. Sonars are good at detecting objects and estimating their distances but provide little information about the localization of a mobile robot in a global reference frame. For this reason, they are often used in conjunction with motion sensors from which odometry can be derived. The most common type of motion sensor is the wheel incremental encoder. While the functioning of incremental encoders is highly reliable at counting wheel revolutions, it suffers from the accumulation of errors leading to odometry drift. Because the linear motion is indirectly derived from the rotation of the wheel, a number of external factors can cause odometry drift. Some of these come from the environment such as the lost of traction with the ground surface. Others may come from the inaccuracy of the wheelbase dimensions or the wheel diameters. In both cases, these errors accumulate with traveled distance and have the detrimental effect of increasing the localization uncertainty.

A possible solution is the use of vision sensors to provide the visual feedback for localization and motion control. Vision-based localization offers a fair alternative to sonars and optical encoders. Camera sensors can be utilized to determine the position and orientation of a robot in a global reference frame. This is possible provided the camera internal projection characteristics and a geometric model of the environment are known a priori. Vision sensors can also collect a wealth of information not available to other

sensors. This information allows for instance to detect and recognize specific navigation landmarks from which localization can either be inferred or updated at regular intervals.

The particular objective of this research is the development of a mobile robotic vehicle and indoor navigational strategies relying on vision as a primary source of sensory percepts. Two problems are addressed specifically: the problem of visual pose estimation and the problem of motion control.

1.3 Approach

The approach to the problem is divided into two distinct segments. The first problem relates to visual sensing and pose estimation using projective geometry techniques. The second addresses the problem of indoor vison-based navigation and motion control.

Visual Pose Estimation Algorithm — The problem of localization consists of finding the position and orientation of the mobile robot with respect to a known world reference. Using robotic vision, the problem essentially becomes one of tracking key visual features or known landmarks contained in the scene being captured. It is here proposed to track points at infinity to recover the orientation. In projective geometry these points are referred to vanishing points. Vanishing points are image locations where parallel lines seem to converge. An image can contain up to three vanishing points, one for each of the principal axis of the camera, however one is usually dominant. By analyzing the movement of the dominant vanishing point contained in an image, it is possible to extract the orientation of a camera with respect to a global reference frame, and that independently of its position. Decoupling the orientation of the camera in space from its translation components constitutes a significant advantage of the proposed approach.

A possible approach to finding the image location of the vanishing point is to extract two line segments[1] parallel to the direction of motion using the Hough transform. The Hough transform [2] is a well known image processing primitive that is used for obtaining a mathematical description of parametrizable features, particularly line segments. Captured images can contain line segments from different sources, many of which do not directly contribute to the dominant vanishing point. However, the line parameter descriptions returned by the Hough transform permits the selection of only two – the dominant ones. The actual coordinates of the vanishing point are then found from the intersection of the two dominant line segments. Because, of the limited spatial resolution of digital images, the uncertainty associated with the line parameters extracted with the Hough transform can be high. To alleviate the detrimental effect on pose estimation, the method presented allows for more lines parallel to the direction of motion to be extracted. In such cases, the coordinates of the vanishing point are obtained by computing a least-square average of all possible line intersections. Assuming the image coordinates of the vanishing point are known, the orientation of the camera can be recovered by performing a three-dimensional Euclidean homography between the coordinates of that point and the coordinates of its world counterpart which is a point at infinity.

From the captured image stream it is also possible to estimate a second component of a robot pose, the lateral position, by tracking known scene feature points that are metrically known. As it will be demonstrated, it is possible to recover the lateral position of the camera in space from the same two line segments that were used in the calculation of the vanishing point

[1] Parallel line segments are referred to natural landmarks when mentioned in the context of world frame structures while features refer to their image frame counterparts.

and their respective projection onto the camera image plane. In essence, the method consists in strategically positioning two hypothetical points in the scene such that the Euclidean distance between them is made equal to the distance separating the parallel line segments. The mathematical descriptions of the line features found with the Hough transform provide the additional constraints required to fully determine a homography problem.

An advantage of this approach, is that tracking line is more reliable than tracking feature points. Feature points can more easily shift outside the field-of-view of the camera than lines. However, the approach requires the identification of the camera projection characteristics commonly referred to the intrinsic parameters. These include the focal length, the image center coordinates, and the pixel aspect ratios. They can be determined from a camera calibration procedure.

Upon finding the orientation and position of the camera in space, the orientation and lateral position of the vehicle can be obtained with a rigid body transformation which transforms the camera coordinates into the coordinates of the vehicle's body given a suitable rotation representation is previously assigned.

The localization problem is fully determined by estimating the third component of the pose, the longitudinal displacement, from the integration of incremental motion using collected odometry data. Two alternatives are considered here. Odometry data can either be directly collected from two incremental encoders mounted on the wheel axles of the mobile robot or derived from the integration of speed measurements collected by two velocity sensors. The strengths and weaknesses of both approaches are assessed experimentally. The pose estimates derived are used as feedback to a pose tracking controller

Pose Tracking Controller — The problem of navigation and motion control is approached by considering a simple visual servoing control problem. For that purpose, the mathematical derivation of a pose tracking controller is presented. The proposed visual servoing control law uses the geometrical properties of Cardinal splines to derive the desired servo input commands. As it is shown, it is possible to vary the spline curvature by properly specifying its endpoint tangency conditions. In this case, the curvature at the origin of the spline is of particular interest as it is used as a system input for controlling the steering velocity of the vehicle body during online navigation.

Implementation — The visual pose estimation algorithm and the pose tracking controller are implemented and tested with AUMER; a robot prototyped as part of the current research. The experiments are conducted in a two-dimensional space virtually free of obstacles. It is here proposed to track corridor ceiling features as natural landmarks. Drop-in tile grid structure, commonly found in office complex, makes a good candidate for the preliminary navigation experiments. Other surfaces containing two distinctive and parallel lines oriented in the direction of motion could also be considered as equally good alternatives. These may include the edge formed by the floor with the walls of a corridor or aisle markers found on the floor of factories and warehouses.

1.4 Scope

The scope of this research includes the development and implementation of a mobile robotic vehicle equipped of a monocular camera and its control system. The adopted control strategy consists of a two-layer hierarchy. The lowest layer of the hierarchy is implemented for dealing with the tracking of

the reference drive-wheel velocities, while the highest level manages visual sensing and pose feedback.

The highest-layer is probably where most of the research efforts are being concentrated as it constitutes the core of the proposed navigational strategy. It integrates the visual pose estimation algorithm and a pose tracking controller. The proposed navigation system and algorithms are evaluated experimentally. In its current implementation, the navigational system provides a minimal set of navigation capabilities. Therefore, navigation is restricted to following rectilinear trajectories with no sharp cornering. The possibility of extending the navigational capabilities to larger-scale autonomous systems is considered by presenting a generic navigational framework.

Because the conclusions of this research may be used by an industrial associated with the project, a significant aspect of this work is to present a solution that is at once practical and cost-effective. An attempt is made to find a solution that is generic, that has wide applicability, and is not restricted to a specific environment. Targeted large-scale applications range from mobile robotic patrol and surveillance to task-dedicated industrial applications.

1.4.1 Summary of contributions

The main contributions of this research project encompass the field of mobile robotics and image analysis. They are stated as follows:

- design and implementation of a mobile robotic vehicle (Chapter 3);
- a complete kinematic/dynamic model of the vehicle (Chapter 4);
- an automated strategy for tracking vanishing points (Chapter 6);
- an accurate visual pose estimation algorithm (Chapter 6);
- goal-directed navigation and motion control (Chapters 6 and 7).

1.5 Organization of the Contents

The remainder of this thesis is as follows: **Chapter 2** gives an overview of the previous work done in mobile robot localization. The discussion concentrates on localization and pose estimation techniques using various types of sensors, such as incremental encoders, magnetic compasses, sonars, and cameras.

Chapter 3 describes the particular robot architecture and its construction. The chapter is composed of two parts. The first part covers the hardware design. It consists of a description of the various mechanical, electrical, and electronical components used in the fabrication of the prototype, including the sensors. The second exposes the details of the control architecture and its software implementation.

Chapter 4 deals with the low-level control system. A dynamic model of the vehicle is derived partially from analytical mechanics and partially from an identification process. The model obtained is then used to synthesize the velocity feedback control law of the robot wheel-drive actuators.

Chapter 5 details the onboard monocular vision system. For the purpose of camera calibrations, a generic mathematical model of the vision system is presented. The parameters of the model are identified from a camera calibration procedure along with a quantitative assessment of lens distortion. A review of pertinent concepts related to lens optics precedes the discussion.

Chapter 6 presents the vision-based navigational system and the current navigational capabilities. It consists of two main components: a visual pose estimation algorithm for position sensing and a pose tracking controller for motion control. The controller utilizes the geometric properties of Cardinal splines for controlling the heading orientation of the vehicle.

Chapter 7 presents indoor navigation results. Regarding localization accuracy, the results of six sensory fusion schemes are compared with the results of the visual pose estimation algorithm. With regards to motion control, the path following performance and responsiveness of the pose tracking controller are evaluated by conducting a set of three navigation experiments. For each of the experiments the controller parameter is varied and its effect on the heading orientation observed.

Chapter 8 concludes with a summary of the main contributions of this work to the field of mobile robotics and image analysis. Promising ideas for future work are also highlighted.

Chapter 2

Literature Review

"Literature is the art of writing something that will be read twice."

– Cyril Connolley

Chapter 2

Literature Review

2.1 Introduction

Mobile robotics is a multidisciplinary research field that relates to two fundamental research areas: localization and motion control. Localization is ubiquitously discussed in the mobile robot literature. It is the process of determining the current position and orientation of a robot relative to its environment from its sensory data [3]. Localization is essential to providing mobility and navigational capability to a robot as it is used retroactively for controlling its motion.

In this chapter, a review of mobile robot navigation schemes is presented. The discussion concentrates on the sensors and methods used for localization. Section 2.2 deals with the more traditional types of sensors and techniques used to measure or infer mobile robot localization. Among others are wheel encoder odometry and ultrasonic ranging. Section 2.3 covers vision-based localization methods. The chapter ends with a short synthesis providing general guidelines for original contributions.

2.2 Localization Methods

Various localization techniques have been proposed in the literature. [4] is an authoritative survey that links sensor hardware to positioning techniques. Positioning techniques can be grouped into two main categories depending on wether they are relative or absolute. Mobile robot navigational strategies usually rely on one of each type [4].

Relative positioning is a localization technique that, in general, uses sensors that do not directly measure the localization variables. Wheel encoders are good examples where the linear displacement of a robot is inferred from its departure point and the rotation of its wheels. A major drawback of relative positioning techniques is that they suffer from accumulating odometry errors.

Absolute positioning is generally achieved with world references that are metrically known in a global coordinate system. Natural landmarks, beacons or man-made artificial markers are examples of absolute references commonly used for providing localization. Using these references, the position of a robot can generally be inferred with good accuracy. A number of sensors can provide information for indoor absolute positioning including compasses, RF beacons, and cameras.

2.2.1 Odometry

Odometry is the most widely used method for obtaining an estimate of the localization of a mobile robot relative to its initial position [4]. Odometry is usually derived from incremental encoders mounted on the wheel drive actuators. A linear relationship in the form of simple geometric equations, exists between the rotation of the wheel and the linear displacement of a robot. These simple geometric equations when considered together form the

kinematic model of the robot. A kinematic model always has some inaccuracies, notably, with respect to the wheelbase and the wheel diameter dimensions. These inaccuracies contribute to systematic localization uncertainties by continuously affecting the measurements. Other sources of error include the loss of traction of the wheel with the ground surface. However, because these are aperiodic in nature, they tend to be less detrimental to wheel odometry.

In odometry-based positioning, any small momentary orientation error will cause a constantly growing position error [4]. For that reason, odometry sensors are often complemented with an independent heading sensor. Orientation can be measured with magnetic compasses, inertial navigation system, or calculated from odometry data [5]–[7]. Magnetic compasses can measures the absolute heading orientation based on the magnetic field of the earth. One advantage of magnetic compasses is that they are not subject to the accumulation of errors. However, the accuracy of compass measurements is affected by surrounding magnetic anomalies. Among these anomalies are magnetic interferences. Magnetic interferences include electromagnetic sources, such as the electrical fields emanating from electrical motors and wireless transceivers [3]. Ferromagnetic structures commonly found in a building complex have also been reported as a significant magnetic disturbance [5]. Magnetic interferences distort locally the magnetic field of the earth, which in turn causes compass readings to deviate from the actual bearing to the magnetic north. As reported by [6], motors may influence compass readings up to 10°. Large ferromagnetic structures are even more detrimental with compass deviation reaching up to $\pm 180°$ [8], making magnetic compasses of limited use for indoor navigation applications. Outdoor robotic navigation systems are less prone to such interferences.

In an interesting approach, Suksakulchai and collaborators [5] suggest taking advantage of the magnetic field disturbance for distinctive places recognition and absolute positioning. They proposed to use magnetic compasses in conjunction with odometry techniques for increasing the reliability of localization by allowing to periodically reset the odometry counts every time a distinctive magnetic signature is encountered. A similar approach is proposed by [3].

Another method of estimating the position of a mobile robot is the inertial navigation system [9]. An inertial navigation system measures the position, velocity, and absolute orientation by measuring the acceleration of an inertial frame. Typically, inertial navigation system uses a combination of gyroscope and accelerometer from which an estimate of velocity and position are obtained by integrating acceleration overtime. The main advantage is that no external sensing is required. However, like wheel encoder odometry, it is subject to the accumulation of error overtime. The drift is particularly important with position as it must be integrated twice. Drift of 1 to 8 cm/s is reported [9].

2.2.2 Ultrasonic Ranging

Another popular mobile robot localization technique is time-of-flight ranging. Time-of-flight ranging systems use the propagation properties of sound or light to determine the range of nearby objects. A common type of ranging system is the sonar. Sonars are cost effective and their signal relatively easy to collect and interpret. They are used in a number of applications. The more common ones are simultaneous map building and localization (SLAM) [10]–[11], and obstacle detection [12].

Common sonar ranging systems are intrinsically highly directional, $\pm 15°$ [4]. While this increases the relative localization accuracy, it limits the

spatial coverage. Several methods have been proposed to overcome this limitation. In [12] it is proposed to use a ring of thirty sonars, equally distributed around the robot base, and fuse the data stochastically. In [6] a different approach is proposed, it uses only one sonar actuated with a servomechanism such that a 180° sweep of the surrounding area is performed at regular time intervals during navigation. The collected data is then used for building an occupancy grid from which the robot's position is recovered with the localization algorithm proposed in [13].

Other successful mobile robot positioning methods have been achieved with global positioning systems (GPS) [14], laser range finder [15], and ultrasonic beacons [16].

2.3 Vision-based Localization

Vision sensors are increasingly utilized in mobile robotic navigation. They are primarily used for obstacle detection [17], visual odometry [18]–[19], or directly in the feedback control loop for visual servoing [20]. Localization strategies that rely on vision as primary sensory input usually requires the visual identification of landmarks or visual features from which the position of the robot can be inferred. The extraction of landmarks brings up several difficulties, the foremost being the reliable identification and extraction of landmarks in environments that are subject to periodic modifications.

[18] proposes the fixation of a known pattern as the robot moves from which the robot's position and orientation are extracted. The method requires a priori knowledge of the pattern dimensions but also requires the extraction of homography features. Extracting homography features in a dynamic environment is nontrivial and many factors can lessen the reliability of the

methods. The targets may become partially occluded or not visually detectable due to varying lighting conditions.

A feature that is not sensitive to occlusion is the vanishing point. Vanishing points are points at infinity where parallel lines seem to converge. They are formed from the perspective projection of three-dimensional world objects onto a two-dimensional image plane of a camera. A vanishing point need not be visible into the image, its location can be calculated from the intersection of the projected parallel line segments converging to it.

Schuster [21] makes use of a dominant vanishing point to recover the orientation of a robot directly in the image domain. It is achieved by performing a planar homography of the point at infinity. As is demonstrated, no metric knowledge of the environment need be known only the projection characteristics of the camera. These include the focal length, the pixel aspect ratio, and the location of the principal point. The principal point, is the point formed by the intersection of the camera principal axis with the image plane. These characteristics can be obtained from a camera calibration procedure [22]–[23]. Because vanishing points are independent of translation, the method proposed in [21] fails to identify a robot's position. One method for identifying a robot's position is optical flow odometry.

Visual odometry is a relatively new approach in robot localization. It is a technique from which a motion estimate is obtained by analyzing the visual structure across a video sequence. One advantage of this method for mobile robot localization is that it is independent of wheel slippage and of the robot's geometry. However, visual odometry has typically been limited to the measurement of incremental linear motion.

Fernandez and Price [18] overcame this limitation by proposing a pseudo-optical flow technique that tracks a number of discrete points from frame to frame with the assumption that the tracked features undergo a constrained affine transformation consisting of a plane rotation and a plane translation. Visual odometry is computationally intensive. Because, optical flow methods typically assume only relatively small change from frame-to-frame, the increased processing time required at high travel speed may become prohibitive to any real-time feedback control. One possible solution to this issue is the use of a dedicated visual odometry subsystem using specialized computing hardware.

In this work, a visual pose estimation algorithm is developed based on the work of [21]. It is proposed to use vanishing points for the recovery of the heading orientation. In addition, the method presented in [21] is extended to recover lateral position simultaneously using planar homographies of ceiling features. The originality of the approach resides in the use of hypothetical correspondences which do not require image feature extraction other than two lines contributing to a dominant vanishing point. The localization problem is fully determined by fusing the pose variables obtained from the visual pose estimation algorithm with wheel encoder odometry. Using this approach, the detrimental effects of odometry drift associated with the estimation of the heading orientation are eliminated without the need for the periodical reset of odometry estimates with other absolute positioning methods.

Chapter 3

Robotic Vehicle Architecture

"One, a robot may not injure a human being, or through inaction, allow a human being to come to harm; Two, a robot must obey the orders given it by human beings except where such orders would conflict with the First Law; Three, a robot must protect its own existence as long as such protection does not conflict with the First or Second Laws."

– Isaac Asimov

Chapter 3

Robotic Vehicle Architecture

3.1 Introduction

This chapter covers the engineering and implementation of a mobile robotic vehicle. Robotic vision is the primary characteristic of the system. Three main aspects are covered: the robot physical attributes, the sensors and onboard processors, and the software control architecture.

First, Section 3.2 gives a description of the hardware components used in the design of the prototype. The characteristics of the sensors and drive actuators that equip the vehicle are given in detail. Section 3.3 describes the software architecture and the different software components developed. In this section, a human-machine interface developed for remotely interacting with the robot is presented. Section 3.4 ends the chapter with a discussion about integration issues and possible improvement to the current architecture.

3.2 Hardware Architecture

The current prototype, named AUMER, is a differentially-driven platform equipped with a vision system for pose estimation and visual servoing. The system includes a radio communication system for external monitoring and control via a human-machine interface. Figure 3.1 shows a picture of the vehicle.

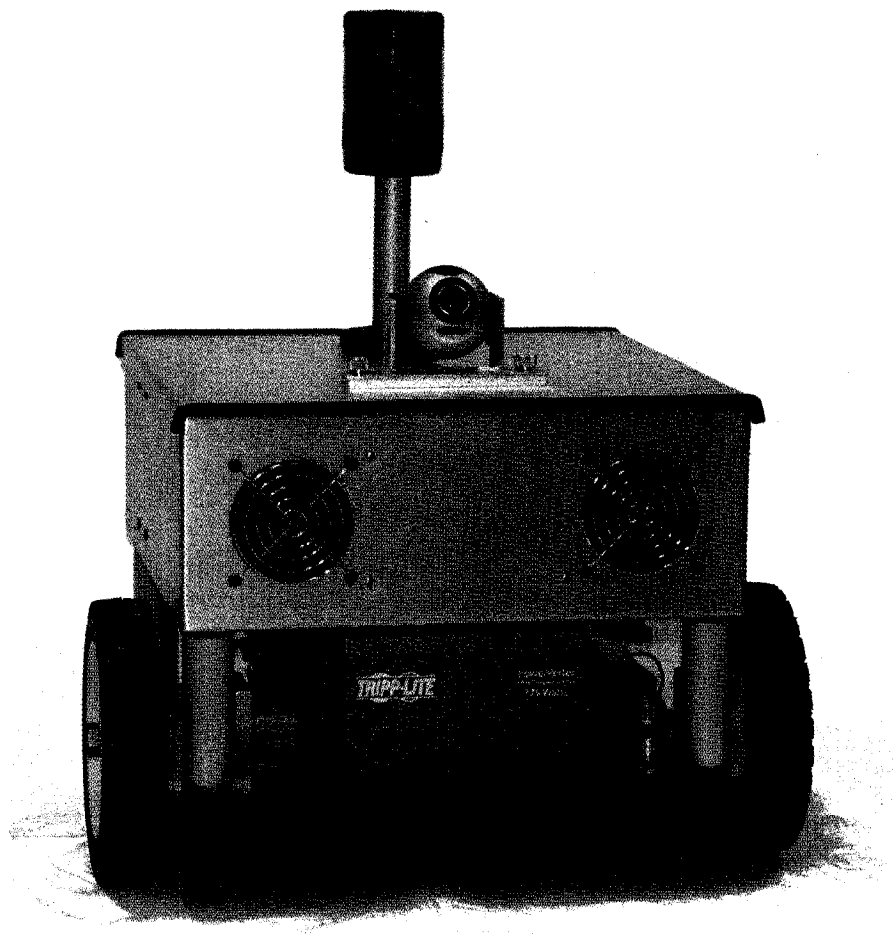


Figure 3.1: AUMER, a differentially-driven mobile robot.

The architecture is divided into six main subsystems: the Chassis, the Power System, the Main Processing Board, the Hardware Controls, the Sensors, and a Base Station. A schematic of the vehicle architecture can be seen in Figure 3.2. A description of the subsystems follows.

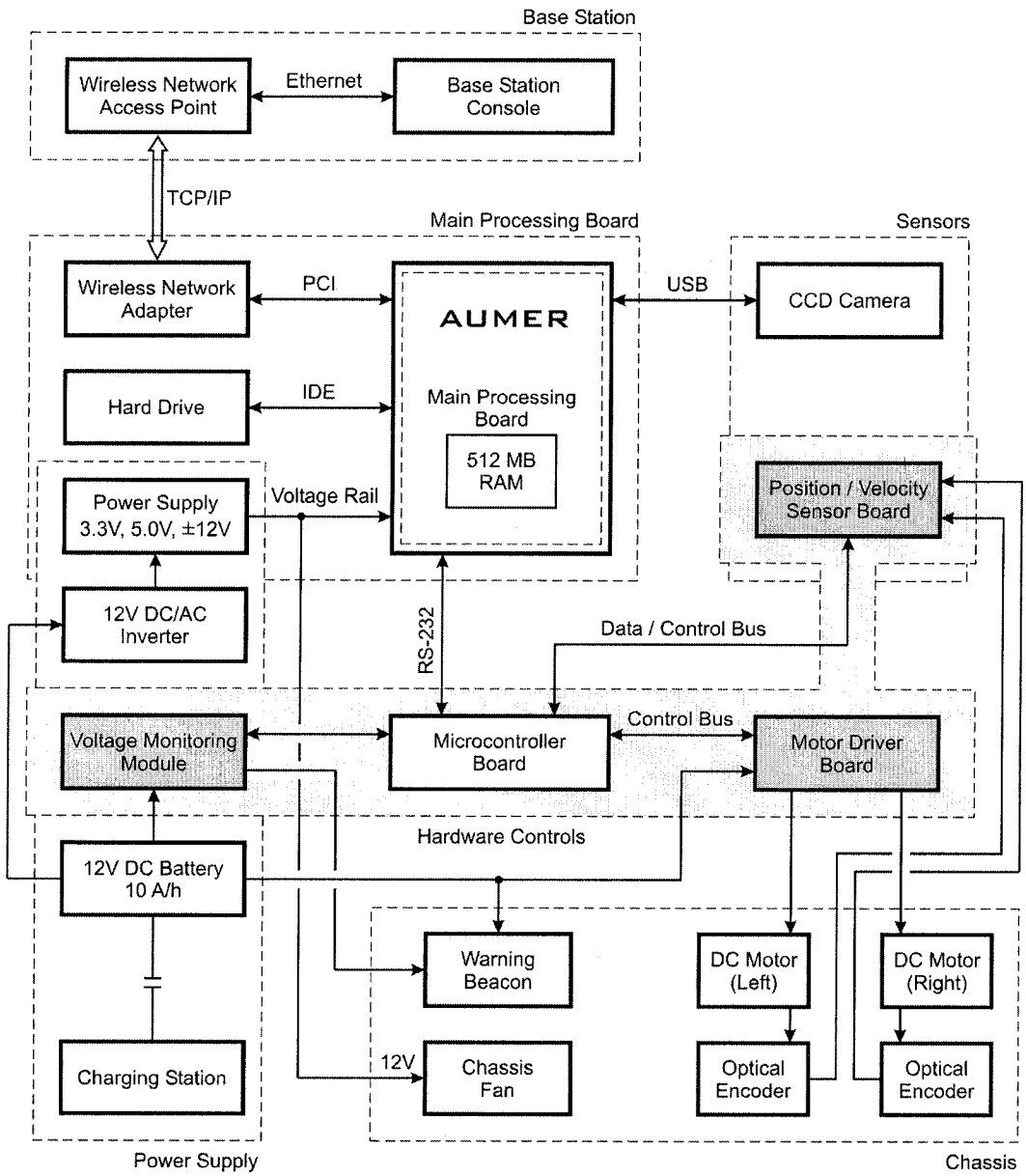


Figure 3.2: Hardware subsystems and components. Shaded blocks indicate custom-engineered modules.

3.2.1 Chassis

The chassis of **AUMER** is composed of two decks. The lower deck is made out of a stiffened aluminum platform of dimension 360 × 360 mm^2 and hosts the Power System. It includes the power supply, the battery pack and the power inverter. Together, these three components account for half the weight of the vehicle, consequently, each component was strategically positioned such that the center of mass of **AUMER** is located close to the geometric center of the vehicle (i.e., the center of rotation). The upper deck is a square aluminum enclosure mounted on top of the lower deck using four 110 mm long spacing risers. The enclosure confines the Main Processing Board, the hard drive, and other electronic components. It also serves as a shield for the electronic components from electromagnetic radiation originating from the DC motors and the power inverter. A 12 VDC fan maintains an air flow of 30 CFM inside the enclosure providing adequate heat dissipation.

Mobility is provided by a pair of wheel assemblies symmetrically mounted at the front end of the lower deck. Each assembly consist of a *Pittman®* *GM9236* permanent magnet DC motor [24], a support bracket, a shaft axle, and a 180 mm diameter wheel. The *GM9236* are high-torque motors (1.7 N·m continuous) geared down to 29.7 r/min with a gear ratio of 65.5:1. Two 40 mm casters fastened underneath the lower deck at the rear end provide tipping stability.

When assembled, the prototype has overall dimensions of 360 × 420 × 360 mm^3 and weighs 20 kg. The primary design characteristics are summarized in Table 3.1. For reference the assembly drawings are included in Appendix A.

Table 3.1: Design characteristics of the prototype.

Characteristics	Descriptions
Overall dimensions:	360 mm × 420 mm × 360 mm
Drive motors:	12 VDC permanent magnet, 1.7 N·m continuous
Gear ratio:	65.5:1
Angular wheel speed:	29.7 r/min (unloaded), 24.7 r/min (loaded)
Travel speed:	0.27 m/s max. (unloaded), 0.22 m/s (loaded)
Workspace radius:	510 mm
Wheel diameter:	178 mm (nominal)
Total weight:	20 kg
Battery life:	0.50 - 0.75 h
Power consumption:	~150 W (continuous), 295.6 W(max)
Power supply:	12 VDC, 120 VAC

3.2.2 Power System

The Power System is composed of three components: a battery, a power inverter, and a regulated power supply. The system acts as the power for both 12 VDC and 120 VAC.

The main power source is a rechargeable 12 VDC sealed lead-acid battery. A DC-to-AC power inverter is used to convert the voltage from the battery into the regulated 120 VAC required by a standard computer power supply for powering the Main Processing Board, the hard drive, and the electronic circuitry. The inverter has an input range of 10.5 VDC to 15.0 VDC that is matched to the battery discharge characteristics. The inverter also provides a number of protection features such as a low battery voltage alarm and a 20 A current overload protection. A standard ATX power supply provides the different voltages required by the Main Processing Board: 3.3 VDC, ±5.0 VDC, and ±12 VDC.

Others peripherals such as the hard drive, the microcontroller board, and sensor boards are powered from the ATX power supply. Power for the DC motors is directly supplied from the battery. Electrical isolation between the direct DC voltage and that for the electronic components was necessary so as to prevent any damage resulting from periodic fly-back voltage emanating from the DC motors. Table 3.2 lists the power consumption of the various electrical components.

Table 3.2: Power consumption.

Components	**Qty**	**Maximum Current (A)**				
		3.3 V	5.0 V	12.0V	12.8V	Power W
Motherboard	1	3	2	0.3		23.5
CPU (2.08 GHz)	1	-	-	-	-	68.3
Chassis fan	1	-	-	0.35	-	4.3
CPU fan	1	-	-	0.2	-	2.4
Memory module (512 MB)	1	-	1.5	-	-	7.5
PCI WLAN adapter	1	-	0.25	-	-	1.3
IDE Hard drive	1	-	0.57	1.5	-	21.0
Freescale M68HC12B32	1	-	0.1	-	-	0.5
CCD USB Camera	2	-	0.5	-	-	5.0
Total (Summary)						**133.8**
Inverter losses (10 %)	-	-	-	-	-	15.0
Power supply losses (30 %)	-	-	-	-	-	64.0
Total DC/AC inverter						**212.8**
DC Motors	2	-	-	-	3	76.8
Warning beacon	1	-	-	0.5	-	6.0
Total battery load						**295.6**
Total current (A)	-	3	4.67	2.85	6	-

The Main Processing Board subsystem accounts for most of the power expenditure with 133.8 W. The next largest load is due to the efficiency losses

of the power supply and the inverter. Adding it all together results in a peak power consumption of 295.6 W. Actual trials with all subsystems powered, the battery lasted more than 30 min. Based on the battery discharge characteristics, it corresponds to a continuous power consumption of about 150 W. It should be noted that the power listed for the motors correspond to the stall or peak torque which should in practice never be reached. The actual power consumption in continuous operation is approximately one third of this value, or about 25.6 W.

3.2.3 Main Processing Board

The Main Processing Board hosts the primary processing power resources for the robotic platform. The processing needs of the autonomous navigation application include: image capture and analysis, high-level navigation control, and background communication with the Base Station.

In the early development stage a design decision was made for using the *Asus*® *A7N8X-VM* motherboard operating a 32-bit *AMD Athlon*™ *XP 2600+* processor with 512 MB RAM. The card measures 245×245 mm^2 and complies with the Micro-ATX form factor. Although, more compact motherboards are becoming standard choices in the field [25], for instance the Mini-ITX (170×170 mm^2) and the proprietary PC/104 (90.1×90.1 mm^2), none of these could accommodate the high end microprocessors required for fast image processing.

The *A7N8X-VM* provides excellent I/O capabilities. It has four USB ports, one parallel port, three PCI slots, and two serial communication ports. One of the serial ports is used to interface with the low-level hardware controller. The second is used for offline development support of the microcontroller firmware. The motherboard is complemented with a 3.5 inch 13.6 GB IDE hard disk drive.

The motherboard provides other hardware functionality, such as built-in audio and video controllers. These are intended for later development of other functionality like speech recognition and robot-human interaction.

3.2.4 Hardware Controls

The hardware control subsystem provides basic control over the platform hardware. It features a *Freescale® M68HC12B32* microcontroller interfacing with three custom-engineered electronic modules designed and built in-house: the Motor Driver Board, the Position/Velocity Feedback Sensor Board, and the Power Monitoring Module.

The benefit of having an embedded microcontroller is that the low-level motion control firmware runs as a real-time deterministic process. The processing load is distributed between the microcontroller and the Main Processing Board which is mainly used for executing the high-level navigation algorithms.

The motor driver circuit includes two *ST L298* dual full-bridge drivers. Since the *L298* has a maximum current rating of 2 A per channel and in order to accommodate the current rating of the motors the outputs of the *L298* were wired in parallel to double the current capacity to 4 A. Each motor is controlled by its own logic inputs tied to dedicated microcontroller I/O's. The H-bridge channel inputs A and B provide full-bidirectional control while the enable input is controlled via a 16 kHz pulse-width modulation signal. Faster circuit response is achieved by controlling the enable input of the bridge, thus avoiding current recirculation across the motor windings. Power delivery to the motor drivers is enabled through software via the human-machine interface. However, a mechanical relay has been given precedence as a safety precaution.

The power monitoring module provides real-time power monitoring. The module uses the voltage level of the battery to estimate the remaining available power. The voltage level is polled at regular intervals using an analog-to-digital converter, and is then displayed at the base station. The module also includes a *MAXIM MAX8211* voltage monitor set to a fixed undervoltage threshold of 11.3 V. Any voltage falling below this threshold is signaled with a warning message. The remote operator is responsible for taking appropriate action. Locally, a LED is lit on the chassis control panel indicating a low battery voltage.

Complete electrical diagrams of the motor drivers and power monitoring module are included in Appendix B. The corresponding printed circuit board layouts (PCB) are also included. For research and development purposes these boards have been kept as separate modules. In future releases, these modules could easily be integrated on a single card interfacing with the main PCI bus of the Main Processing Board.

3.2.5 Sensors

A reasonable level of navigational autonomy is achieved with five different sensors which are used for localization, speed control, and pose tracking. The sensors mounted on the vehicle are:

- one (1) camera sensor;
- two (2) optical encoders; and
- two (2) velocity feedback sensors.

The camera sensor is a *Sony ICX098AK* CCD imager. The sensor has a 640×480 pixels^2 VGA video resolution with a pixel size of 5.6×5.6 μm^2. It can operate at a high capture rate of up to 30 frames per second, and transfer the video stream via a USB port to the Main Processing Board without the

need for a separate frame grabber. The camera sensor is mounted on top of the upper deck enclosure collocated with the robot's geometric center.

As a complement to the vision sensors, odometry data is provided by two *Agilent HEDS-9000* optical encoders assembled onto the motor shafts. Each encoder has three channels and a 512 slots code wheel. The encoder units are capable of providing a resolution in travel distance of 16.6 μm per encoder counts, given a wheel circumference of 559.2 mm. Both optical encoder interface with the *LSI LS7266R1* 24-bit dual axis quadrature counter. The *LS7266R1* inputs and outputs directly maps into the microcontroller CPU's address space. Port A serves as an 8-bit data bus for collecting the 24-bit odometry counters whereas Port B is reserved as the control bus. The direction of rotation is obtained from the *LS7266R1* using the encoders in quadrature mode.

Wheel speed control is achieved with two velocity feedback sensors. The sensors are designed around the *National LM2907* frequency-to-voltage converter. It uses the optical encoder frequency signals as input and linearly converts it into a voltage level ranging from zero to 5 V. The sensors were designed with a maximum encoder frequency input of 16.2 KHz. Provision for sensor calibration is assured with the help of variable resistors soldered onto the printed circuit board. The electrical diagram of the Position/Velocity Feedback Sensor Board and the corresponding PCB layout are included in Appendix B.

3.2.6 Base Station

The base station serves as the wireless communication link between the operator and the high-level control layer of the robotic vehicle. It offers two distinct capabilities. First, it facilitates information transfer from and to the robotic vehicle such as executables, software algorithm, and data collection

files. Second, it allows a remote operator to control, monitor, and supervise the execution of autonomous navigation tasks via the human-machine interface.

The base station hardware includes a PC desktop, a 802.11g compliant wireless access point, and a 54 Mb/s wireless LAN interface mounted on the vehicle's Main Processing Board. A private wireless local area network (WLAN) was established with secure hardware communication using medium access control authentication (MAC). The MAC address of the wireless adapter uniquely identifies the mobile vehicle to the wireless LAN transceiver hub and prevents data snooping and hijacking.

Hardware communication is achieved using the group of internet protocols commonly referred to TCP/IP via the Remote Desktop Connection (RDC). RDC is a client/server service provided by the operating system specifically designed for communicating with a remote computer using TCP/IP protocols. Here, the base station acts as the client where the RDC is essentially used for tunneling into the vehicle computer where the human-machine interface actually resides. The human-machine interface, detailed in Section 3.3.4, provides real-time visualization and basic controls over the robotic vehicle.

3.3 Software Architecture

The control of the vehicle is abstracted into two distinct layers. The low-level control layer is dedicated to motion control and odometry data collection while the high-level control layer is responsible for visual pose estimation and visual servoing. This choice of architecture is logical taking into account the computational latency introduced by the image capture and processing. The control architecture is illustrated in Figure 3.3.

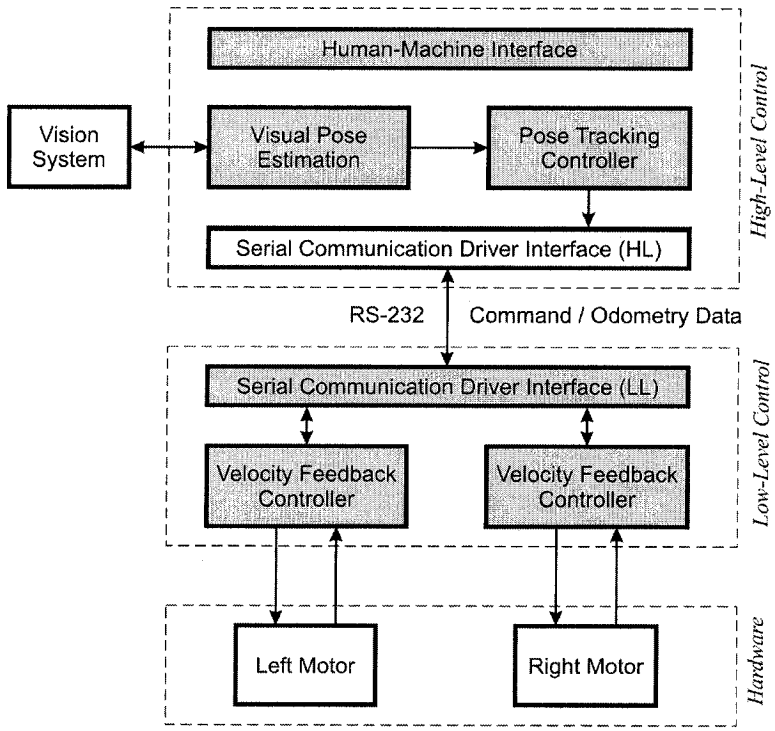


Figure 3.3: Software architecture. Shaded block indicate developed software modules.

3.3.1 Operating System

The operating system of AUMER is *Windows® XP*. *Windows* has been selected because of its multitasking abilities and the variety of built-in services particularly useful to the rapid development of mobile robotic applications. Some of these services include *Remote Desktop Connection* (RDC) and TCP/IP networking as mentioned in the previous section.

Unlike open source software, the *Windows XP* operating system has a rich and well tested programming interface (API) around which the robotic application can evolve. In turns, it allows to focus on the development of navigational algorithms without the burden of developing low-level drivers [25].

On the other hand, *Windows XP* is not intrinsically a real-time operating system. One particular problem it poses concerns the real-time sensor data collection and real-time serial communication between the different control levels. Although the issue can be dealt with to some extent, as was the case in the current software implementation, a number of real-time kernel extensions are commercially available should time determinism and task prioritization be required.

3.3.2 High-level Control

The high-level control layer is implemented on an onboard computer. Two control algorithms were developed using the *C++* programming language. These software programs can be found in Appendix C. The first algorithm relates to visual sensing and pose estimation. The second to visual servoing and pose tracking. Using the visual feedback of the vision system and estimated pose, the pose tracking controller computes the reference velocity commands to the left and right-hand wheels.

Image processing is accomplished through *OpenCV beta 5.0*. *OpenCV* is an open source computer vision library providing a number of image processing primitives particularly useful for robotic applications. These include edge detection, gradient filtering, corner finder, feature tracking, and a variety of transforms. *DirectX 9.0c*, a *Windows* library is also used since it facilitates the image acquisition process by providing a set of simple wrapper functions to access the digital video resources of the Main Processing Board.

Communication between the high and low control levels is accomplished through an RS-232 serial bidirectional communication link at a baud rate of 9 600 bps. The link is used primarily to send the reference velocity commands to the velocity feedback controllers and for collecting odometry data. Aside from visual sensing and motion control, the high-level control layer manages the wireless communication with the base station and the human-machine interface as a separate background process.

3.3.3 Low-Level Control

At this level, two proportional-integral velocity feedback controllers, one for each of the left and right-hand wheel, are responsible for the tracking of the velocity reference commands. These controllers execute on an embedded microcontroller. The embedded firmware of the microcontroller contains more than 2 000 lines of low-level assembly source code compiled with the *68HC12* CPU instruction set. The firmware executes two digital proportional-integral controllers, provides CPU I/O address mapping to the different sensors, and supports full-duplex serial communication (SCI).

The software structure consists of an initialization stage of the *68HC12* peripheral devices followed by the execution of the main routine. The main routine consists of a circular loop that waits for processing internal exceptions as well as external commands from the high-level controller. The handler

routines that respond to external commands support reentrancy and dynamic memory allocations, and are structured such that shared variable conflict is prevented. The code listing is provided in Appendix C along with generic routines for integer arithmetic and data type conversion.

Low-level control commands are channeled via the microcontroller serial communication interface using a simple communication protocol. The protocol defines a list of command interpretable by the robot. Such commands include enabling and setting the speed of the motors among others. The commands can be specified either remotely by an operator via the human-machine interface or directly from the high-level navigation controller. An excerpt of the motor controller command set is listed in Table 3.3.

Table 3.3: Velocity feedback controller command set.

Command	**Flag**	**ID**	**Parameter list**	**Flag**	**Comments**
SPEED	<	S	[short *speed left*][short *speed right*]	>	Set motor speeds.
GAIN	<	G	[byte *proportional*][byte *integral*]	>	Set controller gains.
ENABLE	<	E	[bool *enable* | *disable*]	>	Enable motors.

The command messages are encoded as a fixed length ASCII strings of predetermined semantic. Each of the message packets starts with a message delimiter followed by their unique identifier and the parameter list. Each parameter is coded as a string containing a hexadecimal value. A trailing flag indicates the end of command transmission. Upon reception, the messages are parsed by the microcontroller and the parameters converted into the proper data types as indicated in Table 3.3. Experimentation has demonstrated that successful packet transmission could be achieved at a baud rate of 9 600 bps without the need for an error checking mechanism.

3.3.4 Human-Machine Interface

The human-machine interface provides a way to remotely monitor and control the robot. The graphical interface main window is subdivided into six main areas as can be seen in Figure 3.4. The basic functionalities provided by the interface are discussed next.

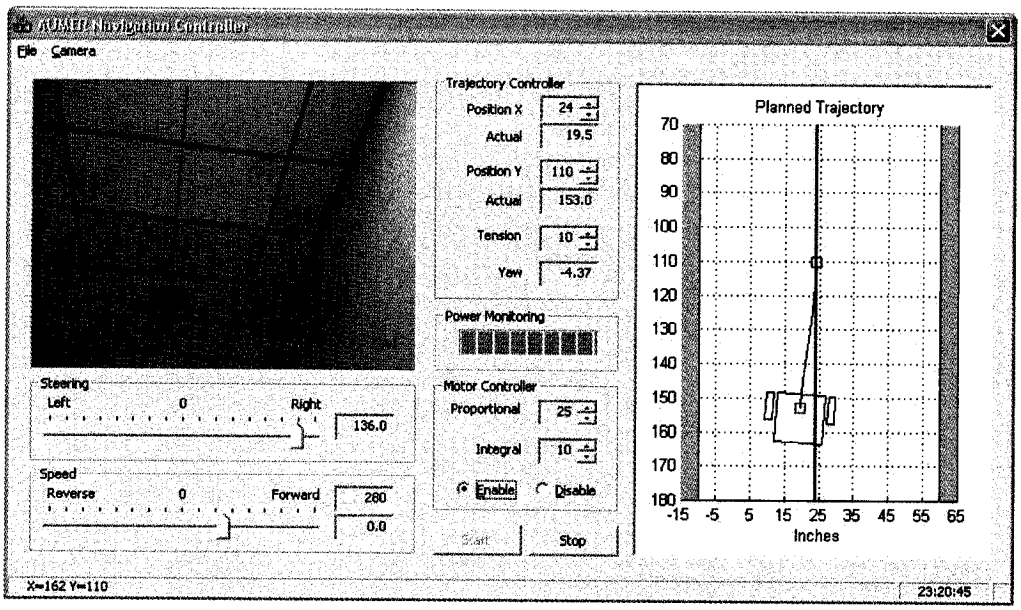


Figure 3.4: Human-machine interface main window.

Manual Mode — Manual control is the default operating mode. In this mode, full control over the robot is achieved via two slider bars located in the subwindow below the video display. One of the sliders is used to specify the steering direction while the other specifies the speed as well as forward or reverse motion. As an alternative to the slider bars, the keyboard arrow keys may be used.

Operational Mode — Pressing the Start button in the main subwindow instantiates the high-level controller in autonomous mode so as to take

control of the robot. In this mode, the robot extracts visual information from the camera sensor and odometry data to navigate in an a priori known indoor environment. Clicking the stop button reverts control to the manual mode.

Video Display — The video displays subwindow shows the video stream captured by the vision system. The images are left unaltered for display. The display is refreshed every 0.5 s.

Map Display — The map display subwindow displays the actual robot pose with respect to the world coordinate frame as well as the computed trajectory to reach the desired path. The map is primarily used to assess the online performance of the visual pose estimation algorithm. The display is refreshed every time new servo command inputs are computed.

Pose Tracking Controller — The pose tracking controls give the user the ability to specify the parameters of the pose tracking controller. The controller algorithm generates a spline between the actual robot's pose and the desired configuration. The user can manually specify the look ahead distance coordinate (Position Y), and adjust the tension parameter of the splines (Tension) via their respective data entry boxes. The tension parameter indicates the smoothness of the spline generated. A value of 0 generates a straight line.

Motor Controller — The motor controller controls allow the user to adjust the speed controller proportional and integral gains. The motors are also enabled or disabled via the radio buttons. Enabling the motors sends a command message to the microcontroller to activate the PWM modules.

Power Monitoring — The power monitoring status bar control displays the voltage level of the battery. This is used to infer the reserve power left.

An empty status corresponds to a critical voltage level of 11.3 V. Any voltage below that level could potentially damage the battery. The reading is updated at regular intervals.

3.4 Discussion

The overall system described is the first prototype developed for the research and development of the navigational algorithm. While the overall robotic vehicle components offer relatively good performance, some improvements could be made in a future release.

Power Efficiency Losses — The Main Processing Board is currently supplied from an AC-to-DC power supply unit. An alternative to this design would have been to use DC-to-DC voltage regulators directly wired to the battery. DC-to-DC power supplies are more efficient than their AC-to-DC counterpart with efficiency of 95 % versus 70 % respectively. On the other hand, selecting a standard AC-to-DC ATX power supply offers more flexibility. In fact, this option allows energizing the vehicle either from the 12 VDC battery or from the 120 VAC utility outlet during testing and development.

Battery Charge Decay — The voltage level of the battery influences the speed of the motors. As a result, the maximum motor speed slowly decays as the battery gradually discharges. Motor speed stability could be enhanced by directly supplying the motors from the power supply which can provide regulated 12 VDC. The power rating of the power supply (350 W) suggests that it can handle the additional load (76.8 W). However, more aggressive means of isolating the electronics from voltage fly-back from the motors would have to be implemented.

Power Density — Although, the overall weight of the platform has been an important design criterion, compromise in terms of cost had to be made. The Power Supply subsystem alone accounts for approximately 10 kg or half the total weight of the vehicle. Batteries with higher energy density, such as a deep-cycle battery, could have been selected, instead, minimizing the effective cost was considered a design objective of highest priority.

Driving Speed Limitation — The preliminary design called for a maximum driving speed of 0.70 m/s. This speed corresponds to the maximum speed of the motors with 30.3 V rated windings. However, the motors are supplied from a 12 VDC battery. The result is that the maximum speed achievable for the moment is limited to 0.27 m/s.

Chapter 4

Modeling and Control

"The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work."

– Johann Von Neumann

Chapter 4

Modeling and Control

4.1 Introduction

The study of kinematics and control of mobile robots has been widely researched, as can be seen by the large body of literature available on the subject [26], [27], [28]. An area where less interest has been given is the characterization of the dynamics. This may in part be attributed to the difficulty in dealing with nonlinearities including: friction, transmission backlash, actuator saturation, or efficiency losses. The great variation in implementations also makes generalization of the problem and the solution thereof, difficult to obtain for a wide class of such vehicles. Gaining knowledge of the exact dynamics, however, provides certain advantages, notably for controller design.

A mathematical model of AUMER and its low-level control system are developed in this chapter. The kinematic equations of the vehicle are first developed in Section 4.2 based on the motion constraints imposed by the hardware. In Section 4.3, Lagrange's formulation is used to derive the differential equations of motion. The model is complemented by using experimentally measured parameters of the robotic vehicle. The result is a

hybrid dynamic model in which part of the dynamics arises from analytical mechanics and part from an identification process. This approach allows the inclusion of prototype nonlinearities and unmodeled effects. In Section 4.4, a model-based technique is employed to synthesize the velocity feedback control law of the wheel drive actuators. Finally, Section 4.5 presents the overall low-level control system and the pose estimation equations.

4.2 Vehicle Kinematics

Figure 4.1 depicts the configuration of Aumer, a differentially-driven robotic vehicle developed in this research. The figure indicates several parameters associated with the robotic vehicle.

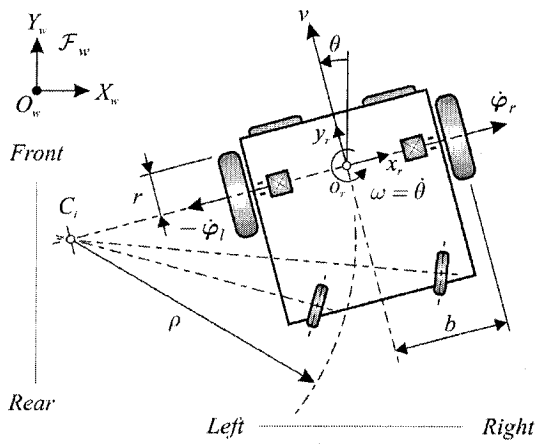


Figure 4.1: Robotic vehicle coordinate system and notations.

The vehicle's generalized coordinates are defined here using five parameters, the localization variables, x and y, the heading angle, θ, defined as the angle between the symmetry axis of the vehicle and the world coordinate frame Y_w-axis, and the angular position of the left and right-hand wheels, φ_l and φ_r respectively. The center of mass C_o is assumed to coincide with the point of intersection of the drive axles with the symmetry axis of the vehicle.

4.2.1 Nonholonomic Velocity Constraints

With reference to Figure 4.1, two kinematic equations and a third constraint on motion can be derived from the differential geometry [29]. It is assumed here that wheel slippage does not occur. The first two equations reflect that assumption. The equation of motion for the left and right wheels are:

$$\dot{x}\cos\theta + \dot{y}\sin\theta - b\dot{\theta} = r\dot{\varphi}_l \tag{4.1}$$

$$\dot{x}\cos\theta + \dot{y}\sin\theta + b\dot{\theta} = r\dot{\varphi}_r \tag{4.2}$$

The third constraint equation is needed to reflect the inability of the vehicle to instantly move in a direction along the wheel axles. This constraint equation is given by:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{4.3}$$

It can be shown that the set of equations of motion of the robotic vehicle is nonholonomic [29]; they cannot be integrated to obtain the constraints on the pose variables[1] x, y, and θ. However, since the interest here is in the rate of change of the position and orientation, the velocity constraint Equations 4.1 to 4.3 can be solved algebraically to obtained the kinematic model of the vehicle. A method for obtaining the pose from these rates is subsequently discussed in Section 4.5.2.

$\dot{\varphi}_l$ and $\dot{\varphi}_r$ are taken as the two controllable independent variables. This coincides with the practical implementation of the prototype. Subtracting Equation 4.1 from 4.2, results in the rate of change of the heading

[1] The tuple (x, y, θ) is commonly referred to as the Cartesian pose.

angle $\dot{\theta}$ (i.e., the nominal angular velocity of the vehicle's geometric center) as follows:

$$\dot{\theta} = \frac{r}{2b}\left(\dot{\varphi}_r - \dot{\varphi}_l\right) \tag{4.4}$$

Adding Equations 4.1 and 4.2, and solving the result with Equation 4.3 yield $\dot{x}$ and $\dot{y}$ as follows:

$$\dot{x} = \frac{r}{2}\cos\theta\left(\dot{\varphi}_r + \dot{\varphi}_l\right) \tag{4.5}$$

$$\dot{y} = \frac{r}{2}\sin\theta\left(\dot{\varphi}_r + \dot{\varphi}_l\right) \tag{4.6}$$

Noting that:

$$\dot{x} = v\cos\theta \tag{4.7}$$

$$\dot{y} = v\sin\theta \tag{4.8}$$

and

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{4.9}$$

The rate of change in position (i.e., the nominal linear velocity of the vehicle's geometric center) can now be expressed as:

$$v = \frac{r}{2}\left(\dot{\varphi}_r + \dot{\varphi}_l\right) \tag{4.10}$$

Using matrix notation, the forward kinematic model of the differentially-driven vehicle becomes:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \Lambda(r,b)\begin{bmatrix} \dot{\varphi}_l \\ \dot{\varphi}_r \end{bmatrix} \tag{4.11}$$

where $\Lambda(r,b) = \begin{bmatrix} r/2 & r/2 \\ -r/2b & r/2b \end{bmatrix}$ is the geometry matrix.

The curvature of the vehicle path is defined as the quotient of the angular velocity of the vehicle and it linear velocity. Dividing Equation 4.4 by 4.9 gives the curvature κ as:

$$\kappa = \frac{\omega}{v} = \frac{1}{b}\frac{\dot{\varphi}_r - \dot{\varphi}_l}{\dot{\varphi}_r + \dot{\varphi}_l} \tag{4.12}$$

Equation 4.12 shows that equal rotational speeds result in a zero curvature, or motion along a straight line, while rotation of the wheels in opposite direction results in infinite curvature, or rotation about the vehicle geometric center, C_o, as shown in Figure 4.1. The radius of curvature, ρ, shown on Figure 4.1 is defined as the reciprocal of the curvature. In terms of vehicle trajectory, it is often more intuitive, to give an explicit specification on the path curvature along with the desired linear velocity. Although, v and κ could theoretically be specified independently, additional constraints arise from the fact that the left and right-hand wheel actuators have limited angular velocities. Consequently, both the maximum linear and angular speed of the robotic vehicle have an upper bound given respectively by

$$|v| \le v_{\max} = \frac{r}{2}\left(\dot{\varphi}_{r,\max} + \dot{\varphi}_{l,\max}\right) \tag{4.13}$$

and

$$|\omega| \le \omega_{\max} = \frac{r}{b} v_{\max} \tag{4.14}$$

Therefore, should the specification of v and κ result in a violation of either Equation 4.13 or 4.14, κ should have precedence over v. In general, following a specified trajectory is a performance objective of higher

importance than tracking the desired speed along the path. In such cases, it is necessary to perform a suitable velocity scaling so as to preserve the curvature corresponding to the nominal specification of v and ω [30]. The desired velocity commands v_d and ω_d are computed by defining [30]:

$$\sigma = \max\left\{\frac{|v|}{v_{\max}}, \frac{|\omega|}{\omega_{\max}}, 1\right\}$$

Then;

if	$\sigma = 1$	then	$v_d = v, \quad \omega_d = \omega$
else if	$\sigma = \lvert v \rvert / v_{\max}$	then	$v_d = \text{sign}(v) v_{\max}, \quad \omega_d = \omega / \sigma$
else			$v_d = v / \sigma, \quad \omega_d = \text{sign}(\omega) \omega_{\max}$

4.3 Vehicle Dynamics

In the previous sections the relation between the wheel drive motion and the motion of the rigid body in Cartesian space was derived. In this section, the focus is on the dynamic behavior of the vehicle. Because of the unique construction of the vehicle, a number of assumptions have to be made. The mass of the vehicle and its rotational inertia are both assumed as lumped parameters. The rotational inertia of each wheel axle components, including the actuators, the wheels and the axle shafts are, however, considered separately. The propelling forces acting on the front wheels are provided by two permanent magnet DC gearmotors mounted on the vehicle lower deck. The rear caster friction forces are not included in the model. An attempt to model the effects of static friction and efficiency losses is presented along with simulations results.

4.3.1 Analytical Model

The analytical model of the robotic vehicle is derived using Lagrange's method. Unlike classical Newtonian dynamics, the method is more efficient in the sense that it only requires the use of energy as a scalar function. The differential equations of motion are derived by applying Lagrange's equation [29]:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = Q_k \tag{4.15}$$

where q_k are the generalized coordinates. For the present case, the generalized coordinates are $q_l = \dot{\varphi}_l$ and $q_r = \dot{\varphi}_r$, the velocities of the left and right-hand wheels respectively. The Lagrangian is defined as:

$$L = T - P$$

where T and P are respectively the total kinetic and potential energy of the system.

Under the assumption that the vehicle undergoes no vertical displacement and moves on a horizontal plane, the kinetic energy of the vehicle body, expressed in terms of the wheel velocity, is given by:

$$T_0 = \frac{1}{2} m_0 v^2 + \frac{1}{2} J_0 \dot{\theta}^2 \tag{4.16}$$

Expressed in terms of the generalized coordinates, Equation 4.16 becomes:

$$T_0 = \frac{1}{2}\left(\frac{r^2}{4} m_0 \left(\dot{\varphi}_r + \dot{\varphi}_l\right)^2 + \frac{r^2}{4b^2} J_0 \left(\dot{\varphi}_r - \dot{\varphi}_l\right)^2 \right) \tag{4.17}$$

where J_o is the vehicle's moment of inertia. Assuming the mass of the left and right-wheel to be identical, their kinetic energies are respectively given by:

$$T_l = \frac{1}{2}\left(\frac{J_m}{n^2} + J_w\right)\dot{\varphi}_l^2 \tag{4.18}$$

$$T_r = \frac{1}{2}\left(\frac{J_m}{n^2} + J_w\right)\dot{\varphi}_r^2 \tag{4.19}$$

where J_m is the polar moment of inertia of the motor rotor, J_w the polar moment of inertia of the wheels, and n the transmission ratio of the gearmotors. The Lagrangian of the system is hence given by:

$$L = \frac{1}{2}\left(\frac{m_0 r^2}{4}\left(\dot{\varphi}_r + \dot{\varphi}_l\right)^2 + \frac{J_0 r^2}{4b^2}\left(\dot{\varphi}_r - \dot{\varphi}_l\right)^2 + \frac{J_m}{n^2}\left(\dot{\varphi}_r^2 + \dot{\varphi}_l^2\right) + J_w\left(\dot{\varphi}_r^2 + \dot{\varphi}_l^2\right)\right) \tag{4.20}$$

Differentiating Equation 4.20 with respect to $\dot{\varphi}_l$ and $\dot{\varphi}_r$ and substituting the result in Equation 4.15 while noting that $q_1 = \dot{\varphi}_l$ and $q_2 = \dot{\varphi}_r$ gives:

$$\left(\frac{m_0 r^2}{4} + \frac{J_0 r^2}{4b^2} + \frac{J_m}{n^2} + J_w\right)\ddot{\varphi}_r + \left(\frac{m_0 r^2}{4} - \frac{J_0 r^2}{4b^2}\right)\ddot{\varphi}_l = Q_l \tag{4.21}$$

$$\left(\frac{m_0 r^2}{4} + \frac{J_0 r^2}{4b^2} + \frac{J_m}{n^2} + J_w\right)\ddot{\varphi}_l + \left(\frac{m_0 r^2}{4} - \frac{J_0 r^2}{4b^2}\right)\ddot{\varphi}_r = Q_r \tag{4.22}$$

Equations 4.21 and 4.22 are the differential equations that represent the motion of the vehicle.

Generalized Forces — The generalized forces acting on the vehicle's body results from the wheel drive actuators. The magnitude of the forces depends on the winding voltages, the back electromotive forces (back emf) and the transmission ratio of the gearmotors. Viscous and static frictions also act as impedance sources. The torque developed by the actuator and referred to the wheels' output shafts, τ, is given by[2]:

$$\tau = \eta \frac{\tau_m}{n} = \frac{\eta}{n} k_t i_a \tag{4.23}$$

where η is the efficiency of the actuators, i_a the armature current, and k_t the torque constant of the motor. The efficiency is considered in the present analysis due to the low transmission ratio of the gearmotors, $n = 1/65.5$, which cause considerable losses that must be accounted for. The torque developed by the actuators, τ_m, is proportional to the current flowing in the armature windings. The voltage equation of the actuators is given by:

$$L_a \frac{di}{dt} + R_t i_a + k_e \frac{\dot{\varphi}}{n} = e \tag{4.24}$$

where L_a is the rotor winding inductance, R_t the rotor winding resistance, k_e the back emf constant, and e the applied armature voltage. The rotor winding inductance, L_a, is generally negligible for DC motors. The armature current can hence be expressed as a function of the applied armature voltage and of its back emf voltage as follows:

$$i_a = \frac{1}{R_t}\left(e - k_e \frac{\dot{\varphi}}{n}\right) \tag{4.25}$$

---

[2] For clarity, the subscripts, *l* and *r*, designating the left and right-hand wheels respectively, are momentarily omitted in the following derivation.

Substituting Equation 4.25 into 4.23 gives:

$$\tau = \frac{\eta k_t}{nR_t}\left(e - k_e \frac{\dot{\varphi}}{n}\right) \tag{4.26}$$

The viscous damping, f_d, of the drive actuator must also be included in the generalized forces as an energy dissipation factor. Its effect, is proportional to the angular velocity of the wheel referred to the output shafts.

Hence, the proposed expressions for the generalized forces for the left and right-hand drive actuators are given as follows:

$$Q = \tau - f_d - \tau_f \tag{4.27}$$

$$Q = \frac{\eta k_t}{nR_t}\left(e_r - k_e \frac{\dot{\varphi}}{n}\right) - \frac{b_m}{n^2}\dot{\varphi} - \tau_f \tag{4.28}$$

where τ_{f_l} and τ_{f_r} account for the static friction torque acting on the drive actuators. Static friction introduces an undesirable nonlinearity in the analytical model.

Substituting Equations 4.28 into 4.21 and 4.22 respectively, and expressing the result in vector-matrix form gives:

$$M\ddot{\varphi}(t) + F_d\dot{\varphi}(t) = K_v e(t) + T_f \tag{4.29}$$

where the inertia matrix $M = \{m_{ij}\}$, $i, j = 1,2$ is defined as:

$$m_{11} = m_{22} = \frac{m_0 r^2}{4} + \frac{J_0 r^2}{4b^2} + \frac{J_m}{n^2} + J_w \tag{4.30}$$

$$m_{12} = m_{21} = \frac{m_0 r^2}{4} - \frac{J_0 r^2}{4b^2} \tag{4.31}$$

The energy dissipation matrix $F_d = \{f_{ij}\}$, $i, j = 1,2$ is diagonal with the principle diagonal entries given by:

$$f_{11} = f_{22} = \frac{1}{n^2}\left(b_m + \frac{\eta k_t k_e}{R_t}\right) \tag{4.32}$$

The voltage-to-torque conversion matrix $K_v = \{k_{ij}\}$, $i, j = 1,2$ is diagonal with the principle diagonal entries given by:

$$k_{11} = k_{22} = k_v = \frac{\eta}{n}\frac{k_t}{R_t} \tag{4.33}$$

Transfer Function Block Diagram — Using the Laplace transform principles, Equation 4.29 can be expressed in terms of separate transfer functions as follows:

$$G_{p_l}(s) = G_{p_r}(s) = \frac{k_p}{T_p s + 1} \tag{4.34}$$

where the time constant of the plant is given by:

$$T_p = \frac{m_{11}}{f_{11}} \tag{4.35}$$

and the dc gain given by:

$$k_p = \frac{k_v}{f_{11}} \tag{4.36}$$

$C(s)$ are the transfer functions of cross-coupled inertia effects and are given by:

$$C_l(s) = C_r(s) = \frac{m_{12}s}{k_v} \tag{4.37}$$

$W(s)$ are torque-to-voltage conversion factors, characteristic of the drive actuators, and are given as follows:

$$W_l(s) = W_r(s) = \frac{1}{k_v} \tag{4.38}$$

Since the multivariable system being dealt with is symmetric, no distinction is made between the left and right channels.

The corresponding transfer function block diagram, expressed in the Laplace domain is shown in Figure 4.2. A possibility of four input-output combinations exists by pairing the multiple inputs to the multiple outputs.

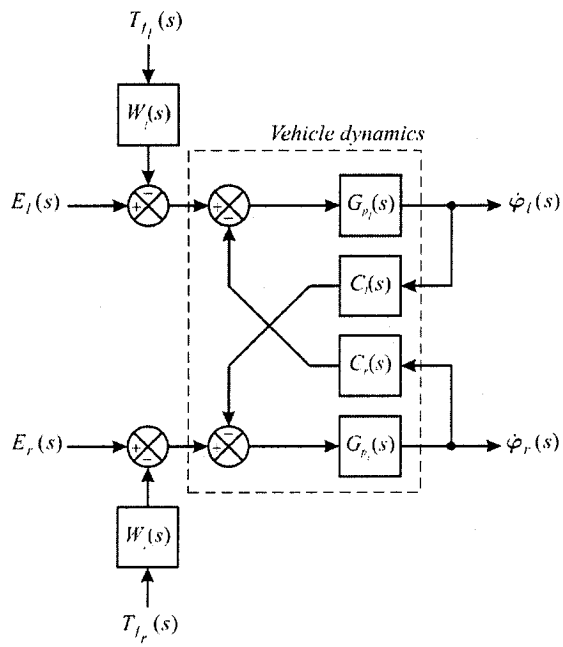


Figure 4.2: Transfer function diagram of the vehicle dynamics.

The inputs $E_l(s)$ and $E_r(s)$ are the Laplace equivalents of the applied armature voltages $e_l(t)$ and $e_r(t)$ respectively. $T_{f_r}(s)$ and $T_{f_l}(s)$ are input load disturbances.

4.3.2 System Identification

The purpose of the system identification experiment is to correlate part of the unknown dynamics of the real vehicle with the analytical nonlinear model derived previously. The unknown dynamics is obtained experimentally by measuring the wheel angular velocities in response to a step input command. The experimental setup for the velocity measurements is shown in Figure 4.3.

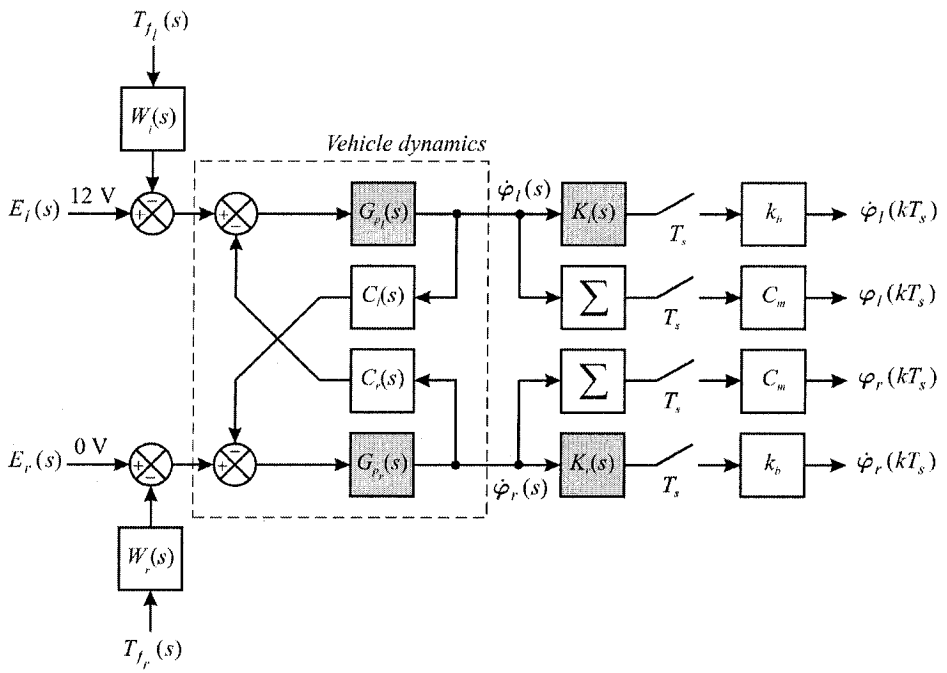


Figure 4.3: Block diagram of the system identification experiment. The shaded blocks indicate the components whose dynamics are identified.

The system identification experiment also aims at identifying the dynamics of the velocity feedback sensors as it influences the selection of the velocity feedback controller gains. The velocity feedback sensors are modeled by first-order transfer functions as follows:

$$K_l(s) = K_r(s) = \frac{k_s}{\tau_s s + 1} \tag{4.39}$$

where k_s is an angular velocity-to-voltage conversion constant, and τ_s is a first-order time constant characteristic of the sensor's circuitry. The conversion constant k_b shown in Figure 4.3 models an analog-to-digital conversion process.

In addition to the velocity feedback sensors, two optical encoders are utilized to collect the angular position of the wheel as a function of time. The summation signs shown in Figure 4.3, represent the discrete pulse integration process as performed by the optical encoder pulse counters, and the constant C_m is the conversion of pulse counts to equivalent angular positions. This constant is calculated as follows [4]:

$$C_m = \frac{2\pi n}{C_e} r \tag{4.40}$$

where r is the radius of the wheels, n the transmission ratio of the gearmotors, and C_e the encoder resolution.

The parameters used for the identification experiment and the subsequent simulations are listed in Table 4.1 to 4.3.

Table 4.1: Physical characteristics of the prototype.

Parameters	Units	Descriptions
$m_0 = 20$	kg	Mass of the prototype
$m_w = 0.25$	kg	Mass of the wheels
$l = 0.36$	m	Length of the vehicle
$r = 0.088$	m	Radius of the wheel
$b = 0.2064$	m	Half the wheelbase dimension
$n = 1/65.5$		Gearmotor transmission ratio

Table 4.2: Velocity feedback sensors and optical encoder parameters.

Parameters	Units	Descriptions
$T_s = 65.6$	ms	Controller sampling time
$k_s = 0.316$	V·s/rad	Velocity feedback sensor constant
$k_b = 256/5$	bit/V	Analog-to-digital conversion factor
$C_e = 512$	pulse/rev	Optical encoder resolution

Table 4.3: Wheel drive actuator parameters.

Parameters	Units	Descriptions
$J_m = 7.06\times10^{-6}$	N·m·s^2	Motor rotor inertia[a]
$b_m = 3.54\times10^{-6}$	N·m·s/rad	Motor viscous damping[a]
$k_t = 0.0582$	N·m/A	Motor torque constant[a]
$k_e = 0.0582$	V·s/rad	Motor back emf constant[a]
$R_t = 3.91$	Ω	Motor winding resistance[a]
$e_{max} = 12$	V	Maximum winding voltage
$\dot{\varphi}_{nom} = 3.16$	rad/s	Nominal angular velocity (unloaded)

[a] Source: *Pittman®* *GM9236* specification sheet [24].

4.3.3 Velocity Feedback Sensor Dynamics

The first identification experiment was carried out to characterize the dynamic behavior of the velocity feedback sensors. Identification of the sensors dynamics was achieved by applying a step input of $E_l(s) = 12$ V to the left-hand wheel drive actuator and by measuring simultaneously the wheel angular velocities with the velocity sensors and the angular positions with the optical encoders. Differentiating the optical encoder signals with respect to time allows one to obtain a duplicate measurement of the wheel angular velocity assuming it is free of any time lag.

The time constant of the sensor is determined by filtering the duplicate velocity signal obtained from the optical encoder with a discrete equivalent of Equation 4.39 until it matched the velocity measurements collected with the velocity sensors. Because of the discrete nature of the measurements, the sensor continuous time constant is recovered as follows.

The Laplace transfer function of the velocity feedback sensors is given by:

$$K(s) = \frac{\dot{\varphi}'(s)}{\dot{\varphi}(s)} = \frac{k_s}{\tau_s s + 1} \tag{4.41}$$

where $\dot{\varphi}'(s)$ is the filtered signal and $\dot{\varphi}(s)$ the measurements. The time domain equivalent of Equation 4.41 is given by:

$$\tau_s \frac{d\dot{\varphi}'(t)}{dt} + \dot{\varphi}'(t) = k_s \dot{\varphi}(t) \tag{4.42}$$

Equation 4.42 can be discretized using the backward rectangular rule approximation:

$$\frac{d\dot{\varphi}'(t)}{dt} \approx \frac{\dot{\varphi}'_k - \dot{\varphi}'_{k-1}}{T_s} \tag{4.43}$$

where T_s is the sampling time. Substituting Equation 4.43 into 4.42 yields:

$$\tau_s \frac{\dot{\varphi}'_k - \dot{\varphi}'_{k-1}}{T_s} + \dot{\varphi}'_k = k_s k_b \dot{\varphi}_k \tag{4.44}$$

where

$$\dot{\varphi}_k = \frac{\varphi_k - \varphi_{k-1}}{T_s}$$

is a duplicate velocity measurement obtained from the optical encoder signal. The conversion constant k_b take into account the effects of the analog-to-digital conversion and has units of bit/V. Rearranging Equation 4.44 gives:

$$\dot{\varphi}'_k = \left(\frac{\tau_s}{\tau_s + T_s}\right)\dot{\varphi}'_{k-1} + \left(\frac{T_s}{\tau_s + T_s}\right)k_s k_b \dot{\varphi}_k \tag{4.45}$$

Alternatively, Equation 4.45 can be expressed in a standard low-pass filter form as follows [31]:

$$\dot{\varphi}'_k = \alpha_s \dot{\varphi}'_{k-1} + (1-\alpha_s)k_s k_b \dot{\varphi}_k \tag{4.46}$$

where

$$\alpha_s = \left(\frac{\tau_s}{\tau_s + T_s}\right)$$

is the low-pass filter constant. From this constant, an explicit expression for the velocity sensor continuous time constant is obtained as follows:

$$\tau_s = \left(\frac{\alpha_s}{1-\alpha_s}\right)T_s \tag{4.47}$$

Figure 4.4 shows the experimental results. The figure shows the angular velocity of the left-hand wheel as sampled by the velocity feedback sensor and the duplicate velocity measurement as obtained by differentiating the optical encoder signal. Agreement between both velocity measurements was achieved by filtering the duplicate velocity measurement using Equation 4.46 with a filter constant value of $\alpha_s = 0.45$ which corresponds to a continuous time constant of $\tau_s = 53.6$ ms.

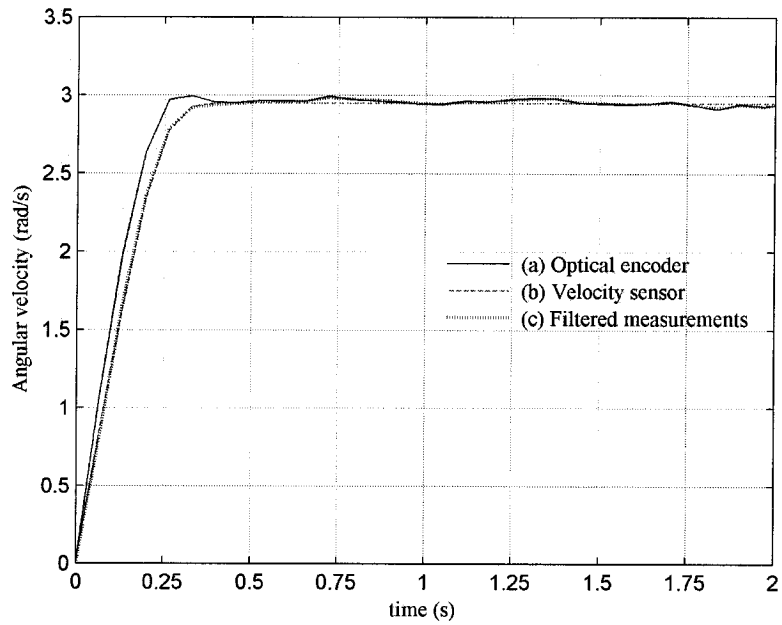


Figure 4.4: Identification of the velocity feedback sensor dynamics. (a) differentiation of the optical encoder signal with respect to time, (b) velocity feedback sensors measurements, and (c) filtered signal obtained in (a) with $\alpha_s = 0.45$.

4.3.4 Prototype Dynamics

The second identification experiment was carried out to characterize the dynamic behavior of the prototype. It was performed by placing the vehicle on a flat open-space and applying a constant input voltage of $E_l(s) = 12$ V to the left-hand wheel drive actuator, while keeping the right-hand wheel drive actuator input voltage set to $E_r(s) = 0$ V. Both controller feedback loops were kept open for the duration of the velocity measurements. The angular velocities of both driving wheels were calculated by differentiating the optical encoder signal with respect to time such that the time lag introduced by the velocity feedback sensors does not bias the identification of the prototype dynamics. The response curves obtained for both the left and right channels are shown in Figure 4.5. The actual parameter values used for the simulation are given in Tables 4.1 and 4.3.

The figure shows simulation results for the system with an input load disturbance of $T_f = 0$ N·m since its magnitude is unknown as of yet, and a maximum gearmotor efficiency of $\eta = 46$ %. The maximum wheel angular velocity reached is $\dot{\varphi}_{max} = 2.94$ rad/s which corresponds to the maximum applied voltage.

The response curves obtained suggest that the system contains two types of nonlinearities. The first is the static friction torque, τ_f. The static friction torque depends mostly on the magnitude of the vehicle load and of the coulomb friction of the drive actuators. Its effect can be seen by the large dc gain offset at steady-state between the two responses. The second, less obvious, is due to the efficiency losses of the drive actuators which varies as a function of the rotational speed but has been assumed constant. The efficiency losses explain the lag observed in the transient portion of the open-loop response of the actual prototype when compared to the response of the

analytical model. Modeling the effects of these efficiency losses is discussed in the next section.

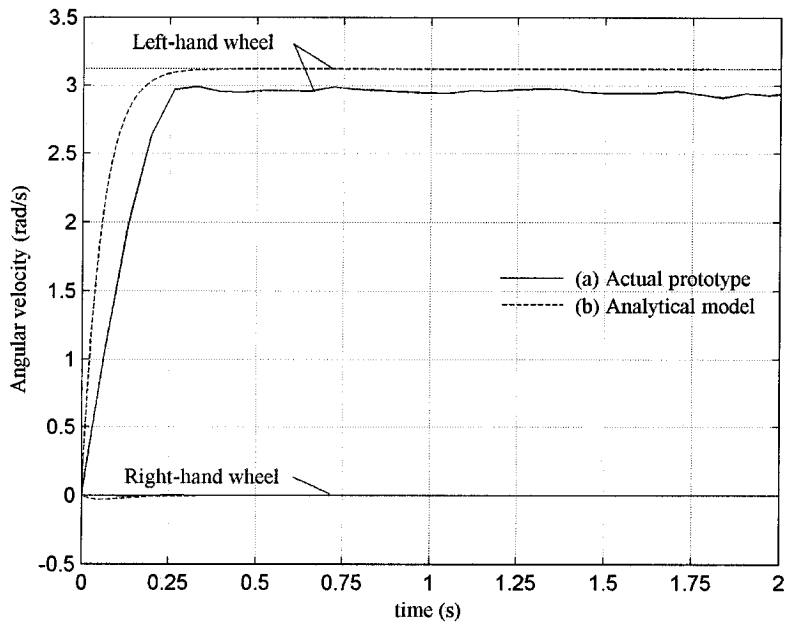


Figure 4.5: Open-loop response to a step input of 12 V applied to the left-hand wheel drive actuator. (a) the actual prototype, and (b) the analytical model given by Equation 4.29 with the parameters $\tau_f = 0$ N·m and $\eta = 46$ %.

Another observation can be made regarding the cross-coupled dynamics. The interaction between the left and right-hand channels is minimal; this is primarily due to the low transmission ratio of the gearmotors. Therefore, the multiple input/multiple output (MIMO) system will be treated as two identical single input/single output (SISO) systems for the purpose of system identification.

Efficiency Losses — The analytical model takes into accounts the maximum efficiency of the wheel drive actuators as per the manufacturer's specifications [24]. However, the efficiency of the drive units is a function of the rotational speed. From the efficiency curves of the drive units [24], it is observed that the efficiency has a linear relationship with the rotational speed of the form:

$$\eta(\dot{\varphi}) = a\dot{\varphi} \tag{4.48}$$

for almost the entire speed range. Neglecting the curvilinear portion of the curve, a proportionality constant, a, can be calculated from the graph. However, substitution of Equation 4.48 into the differential equation of motion, Equation 4.29, introduces an additional nonlinearity, and thus the equation cannot be directly used for classical controller design.

To deal with the nonlinearity resulting from the efficiency effects, it is proposed to use an average efficiency over the entire speed range as opposed to using the maximal efficiency of the drive units. An estimate can be determined by averaging the area under the efficiency curves of the drive units. The result is given by:

$$\overline{\eta} = \frac{\eta_{\max}}{2} \tag{4.49}$$

Figure 4.6 shows the open-loop response of the analytical model after substitution of Equation 4.49 into 4.29.

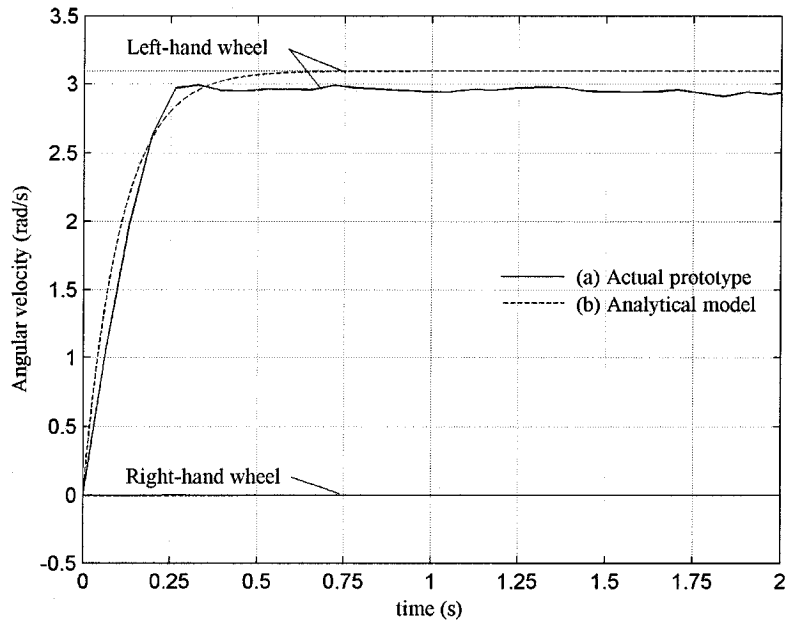


Figure 4.6: Open-loop response to a step input of 12 V applied to the left-hand wheel drive actuator. (a) the actual prototype, and (b) the analytical model given by Equation 4.29 with the parameters $\tau_f = 0$ N·m and $\overline{\eta} = 23$ %.

Figure 4.6 shows that decreasing the efficiency has the effect of decreasing the speed of response in the transient portion. As a result, the improved model now fits the actual process behavior in the low speed range with only minor discrepancies. The cross-coupling effect between the left and right-hand wheel input/output channel has also vanished. Still, the response exhibits a steady-state offset which arises from the static friction torque. Modeling this effect is investigated next.

Static Friction — The most common model for describing the effects of static friction is the Coulomb friction model. Other friction models are proposed in the literature [32] but are not relevant for this discussion. In the coulomb friction model, the force due to static friction is constant in magnitude, but its direction is opposite to that of the relative velocities between the bodies in contact. The effect of the friction force can be modeled by a constant voltage drop that reduces the effective voltage across the drive actuator windings.

For the case of the prototype vehicle, the magnitude of the static friction force is proportional to its mass. It takes the form of a constant torque that must be overcome before any movement of the vehicle can take place. Analyzing the prototype response to an input signal, the magnitude of the lumped friction force for the entire system can be obtained. Since the interest here is in the overall energy dissipation, the following model is proposed:

$$\tau_f = (\beta - 1)\frac{b_m}{n^2}\dot{\varphi}_{\max} \tag{4.50}$$

were $\beta \leq 1$ is the coefficient of equivalent damping, and $\dot{\varphi}_{\max}$ the steady-state velocity corresponding to the maximum voltage, $e_{\max}$, applied during the open-loop identification experiment. Considering Equation 4.28 at steady-state, the following equality is obtained:

$$\frac{1}{n^2}\left(b_m + \frac{\overline{\eta}\, k_t k_e}{R_t}\right)\dot{\varphi} = \frac{\overline{\eta}\, k_t}{nR_t}e - \tau_f \tag{4.51}$$

Substituting Equation 4.50 into 4.51 yields:

$$\frac{1}{n^2}\left(\beta b_m + \frac{\overline{\eta}\, k_t k_e}{R_t}\right)\dot{\varphi}_{\max} = \frac{\overline{\eta}\, k_t}{nR_t} e_{\max} \tag{4.52}$$

where βb_m is the equivalent damping of the system. Equation 4.52 can be solved explicitly for β which results in:

$$\beta = \frac{n^2}{b_m}\left(\frac{\overline{\eta}\, k_t}{nR_t}\frac{e_{\max}}{\dot{\varphi}_{\max}} - \frac{1}{n^2}\frac{\overline{\eta}\, k_t k_e}{R_t}\right) \tag{4.53}$$

The coefficient of equivalent damping, β, is introduced in the system model by modifying Equation 4.32 as follows:

$$f_{11} = f_{22} = \frac{1}{n^2}\left(\beta b_m + \frac{\overline{\eta} k_t k_e}{R_t}\right) \tag{4.54}$$

Consequently, the equivalent voltage drop caused by the frictional torque is given by:

$$e_f = \frac{nR_t}{\overline{\eta} k_T}\tau_f \tag{4.55}$$

Substituting the parameter values in Equation 4.50, 4.53, and 4.55 the following parameters are calculated: the coefficient of increase damping $\beta = 3.97$, the static friction torque $\tau_f = 0.134$ N·m, and the corresponding voltage drop caused by the static friction $e_f = 0.597$ V.

Figure 4.7 shows the simulation results for the system taking into account the identified system parameters. The model shows a remarkable agreement with the open-loop response obtained for the prototype. The settling time of the analytical model is 0.5 s compared to 0.35 s obtained with the actual prototype.

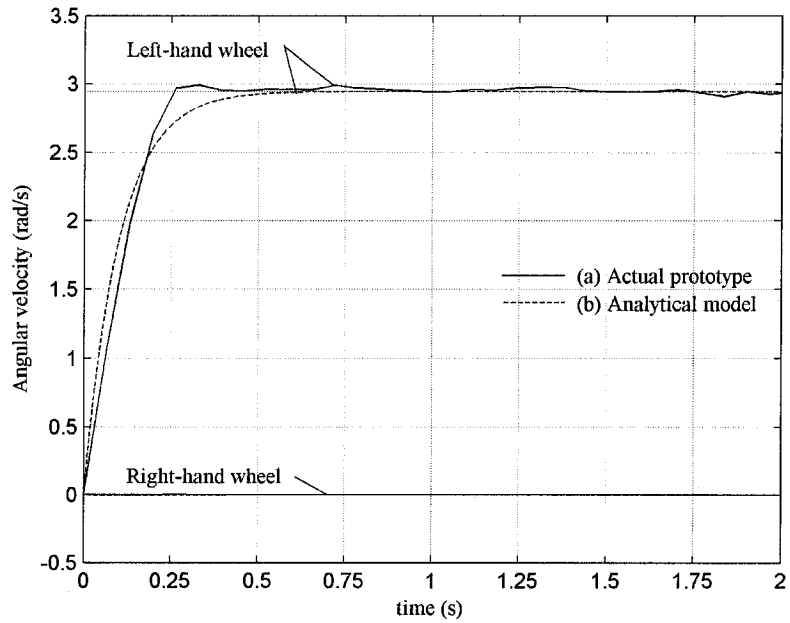


Figure 4.7: Open-loop response to a step input of 12 V applied to the left-hand wheel drive actuator. (a) the actual prototype, and (b) the analytical model given by Equation 4.29 with the parameters $\tau_f = 0.134$ N·m and $\overline{\eta} = 23$ %.

4.4 Low-Level Controller Design

In this section a proportional-integral (PI) velocity feedback control law is presented. The control law is derived based on the decoupled dynamics of the vehicle where each wheel drive actuator is treated separately. The decoupling is achieved by eliminating the off-diagonal elements of the inertia matrix of Equation 4.29. Because the system is symmetric and the cross-coupling effects are negligible, the system is treated as two identical control loops. Again, no distinction is made between the left and right controller. The velocity feedback control law derived will be used to control the angular velocity of both controllers.

For designing the controllers, the Direct Synthesis (DS) approach is used [33]. This approach is preferred to a full-state feedback controller design approach, mainly for the resulting ease of tuning. Using full-state feedback would require a set of four parameters to be specified; the desired closed-loop poles. In contrast, decoupling the dynamics and using direct synthesis, only requires the specification of one parameter, (i.e., a time constant). Finally, the control law is presented in the form of a backward difference equation.

4.4.1 Control law Synthesis

The velocity feedback controller is implemented digitally using the backward rectangular rule.

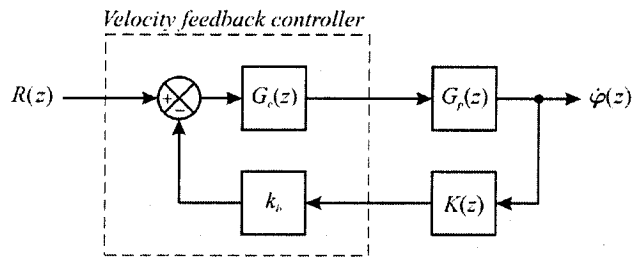


Figure 4.8: Decoupled velocity feedback control loop.

With reference to the block diagram of Figure 4.8, the decoupled closed-loop transfer function is given by:

$$\frac{\dot{\varphi}(z)}{R(z)} = \frac{G_c(z)G_p(z)}{1+k_b G_c(z)G_p(z)K(z)} \tag{4.56}$$

$G_c(z)$ can be directly solved for, and its transfer function is obtained as follows:

$$G_c(z) = \frac{1}{k_b G_p(z)K(z)} \frac{\dot{\varphi}(z)}{R(z)} \left(1 - \frac{\dot{\varphi}(z)}{R(z)}\right)^{-1} \tag{4.57}$$

Assuming the desired closed-loop response is in a form given by:

$$\frac{\dot{\varphi}(z)}{R(z)} = G_d(z)K(z) \tag{4.58}$$

where $G_d(z)$ is the desired first-order response and $K(z)$ is the velocity sensor transfer function. Using the backward rectangular rule approximation given by:

$$s \approx \frac{1-z^{-1}}{T_s}$$

the discrete form of the desired first-order response is given by:

$$G_d(z) = \frac{T_s}{T_s + \lambda\left(1 - z^{-1}\right)} \tag{4.59}$$

where the time constant λ is referred to the controller tuning parameter.

For the purpose of controller design the velocity sensor transfer function given by Equation 4.39 is approximated by a pure time delay equivalent to one sampling period. This is possible because the following two approximations are recognized:

$$K(s) = \frac{k_s}{\tau_s s + 1} \approx k_s e^{-s\tau_s} \tag{4.60}$$

and

$$e^{-s\tau_s} \approx e^{-sT_s} \tag{4.61}$$

where $\tau_s = 56.5$ is approximately equal to one sample time delay since $T_s = 65.6$. Substituting 4.61 into 4.60, and noting that $z^{-1} = e^{-sT_s}$, then the discrete transfer function of the velocity sensors is obtained as:

$$K(z) \approx z^{-1} \tag{4.62}$$

The discrete equivalent of the plant transfer function, Equation 4.34 is given by:

$$G_p(z) = \frac{k_p T_s}{T_s + T_p\left(1 - z^{-1}\right)} \tag{4.63}$$

Substituting Equation 4.62 and 4.63 into the controller transfer function, Equation 4.57, gives:

$$G_c(z) = \frac{\dfrac{T_s}{k_p k_s k_b\left(\lambda + T_s\right)} + \dfrac{T_p}{k_p k_s k_b\left(\lambda + T_s\right)}\left(1 - z^{-1}\right)}{1 - z^{-1}} \tag{4.64}$$

Simplifying Equation 4.64 results in a compact form given by:

$$G_c(z) = \frac{U(z)}{E(z)} = \frac{k_i + k_c\left(1 - z^{-1}\right)}{1 - z^{-1}} \tag{4.65}$$

where the gains k_c and k_i are defined respectively as:

$$k_c = \frac{T_p}{k_p k_s k_b\left(\lambda + T_s\right)}, \quad k_i = k_c \frac{T_s}{T_i}$$

and $T_i = T_p$.

Taking the inverse Z-transform of Equation 4.65 using:

$$Z^{-1}\{E(z)\} = e(kT_s) \quad \text{and} \quad Z^{-1}\{z^{-1}E(z)\} = e(kT_s - T_s)$$

the backward difference equation of the controller is obtained as follows:

$$u(kT_s) = u(kT_s - T_s) + k_c\left[e(kT_s) - e(kT_s - T_s)\right] + k_i e(kT_s) \tag{4.66}$$

Using short notation Equation 4.66 becomes:

$$u_k = u_{k-1} + k_c\left(e_k - e_{k-1}\right) + k_i e_k \tag{4.67}$$

where the subscript k indicates the sampling time. Equation 4.67 is the actual proportional-integral velocity feedback control law that is implemented in assembly language for controlling the speed of both wheel drive actuators. The program code is included in Appendix C.

4.5 MultiLoop Description

The multiloop block diagram representation of the robotic vehicle is shown in Figure 4.9.

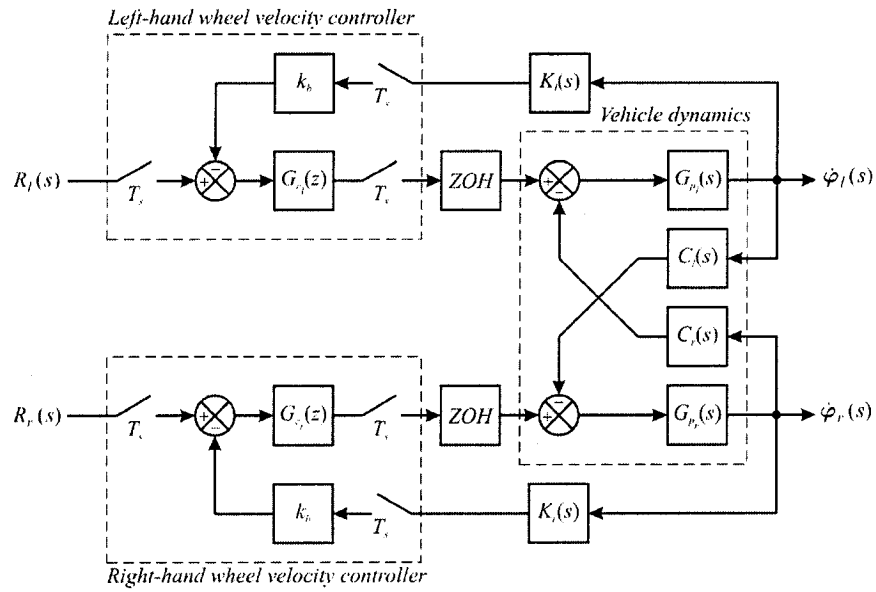


Figure 4.9: Multiloop control representation of the robotic vehicle dynamics.

With reference to the above figure and using the properties of block diagram reduction, the feedback control loop expressions for the wheel angular velocity can be derived in vector-matrix notation form as follows:

$$\begin{bmatrix} \dot{\varphi}_l(z) \\ \dot{\varphi}_r(z) \end{bmatrix} = \begin{bmatrix} G_{11}(z) & G_{12}(z) \\ G_{21}(z) & G_{22}(z) \end{bmatrix} \begin{bmatrix} R_l(z) \\ R_r(z) \end{bmatrix} \tag{4.68}$$

where the elements $G_{11}(z)$, $G_{12}(z)$, $G_{21}(z)$, and $G_{22}(z)$ are the transfer functions between the inputs $R_l(z)$, $R_r(z)$ and the outputs $\dot{\varphi}_l(z)$, $\dot{\varphi}_r(z)$ combinations.

These transfer functions are given by:

$$G_{11}(z) = \frac{\dot{\varphi}_l(z)}{R_l(z)} = \frac{G_{c_l}(z)H_l G_{p_l}(z)\Delta_r(z)}{\Delta(z)} \tag{4.69}$$

$$G_{12}(z) = \frac{\dot{\varphi}_l(z)}{R_r(z)} = -\frac{G_{c_r}(z)H_r G_{p_r} C_l G_{p_l}(z)}{\Delta(z)} \tag{4.70}$$

$$G_{21}(z) = \frac{\dot{\varphi}_r(z)}{R_l(z)} = -\frac{G_{c_l}(z)H_l G_{p_l} C_r G_{p_r}(z)}{\Delta(z)} \tag{4.71}$$

$$G_{22}(z) = \frac{\dot{\varphi}_r(z)}{R_r(z)} = \frac{G_{c_r}(z)H_r G_{p_r}(z)\Delta_l(z)}{\Delta(z)} \tag{4.72}$$

and

$$\Delta(z) = \Delta_l(z)\Delta_r(z) - \Delta_{lr}(z) \tag{4.73}$$

The denominator $\Delta(z)$ is the characteristic equation that determines the dynamic behavior of the cross-coupled closed-loop systems. The first two terms on the right-hand side of Equation 4.73 account for the dynamics associated with the decoupled left and right-hand input-output channels and are given by:

$$\begin{aligned}\Delta_l(z) &= 1 + k_b G_{c_l}(z) Z\left\{H_l(s)G_{p_l}(s)K_l(s)\right\} \\ &= 1 + k_b G_{c_l}(z)H_l G_{p_l} K_l(z)\end{aligned} \tag{4.74}$$

$$\begin{aligned}\Delta_r(z) &= 1 + k_b G_{c_r}(z) Z\left\{H_r(s)G_{p_r}(s)K_r(s)\right\} \\ &= 1 + k_b G_{c_r}(z)H_r G_{p_r} K_r(z)\end{aligned} \tag{4.75}$$

The last term of Equation 4.73, $\Delta_{lr}(z)$, is characteristic of the dynamic effects due to the cross-coupling of the controlled variables $\dot{\varphi}_l$ and $\dot{\varphi}_r$ and is given by:

$$\begin{aligned}\Delta_{lr}(z) &= Z\left\{C_l(s)G_{p_l}(s)D_l(s)C_r(s)G_{p_r}(s)D_r(s)\right\} \\ &= C_l G_{p_l} D_l C_r G_{p_r} D_r(z)\end{aligned} \tag{4.76}$$

An exact model for the sample and hold power modulation process is used in the derivation of the discrete transfer function expressions, this model is given by [31]:

$$H(s) = \frac{1-e^{-sT_s}}{s} \tag{4.77}$$

It should be noted that since the off-diagonal elements $G_{12}(z)$ and $G_{21}(z)$ are non-zero, a set point change in one loop causes both wheel angular velocities to change. Also, since the model transfer functions all have the same denominator, $\Delta(z)$, the stability of the system depends on both controller gains. However, for the prototype vehicle $G_{12}(z)$ and $G_{21}(z)$ are negligibly small. Consequently, the system can be looked at as decoupled and the effect of one control loop on the other as an external disturbance.

4.5.1 Low-level Control System

The low-level control system model encapsulates the kinematics and the closed-loop dynamics of the vehicle. It runs on the embedded controller used to interface with the wheel drive actuator. This allows the controllers to respond promptly to changes in paths by adjusting the left and right-hand wheel angular speeds without being affected by the high-level control algorithms running on the mainboard processor.

The high-level control algorithm uses visual information supplied by the camera sensor to determine the desired reference commands for the low-level controllers, namely v_d and ω_d which are respectively the driving and steering velocities. The kinematic model acts at the interface as a mean to convert the specified set points into directly controllable variables $\dot{\varphi}_l$ and $\dot{\varphi}_r$. Figure 4.10 shows the schematic representation of the low-level control system model.

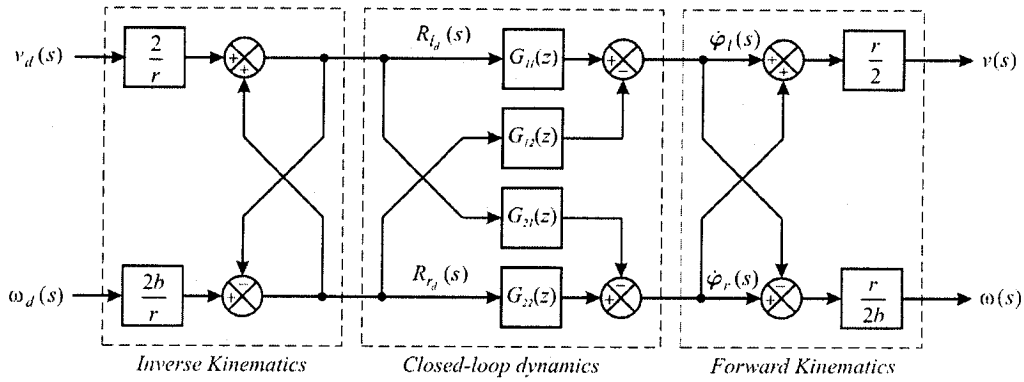


Figure 4.10: Model of the low-level control system.

Considering the output variables of the system to be v and ω, the block diagram of Figure 4.10 can be concatenated in matrix form to obtain the following expression:

$$\begin{bmatrix} v(s) \\ \omega(s) \end{bmatrix} = \Lambda(r,b)G(z)\Lambda(r,b)^{-1}\begin{bmatrix} v_d(s) \\ \omega_d(s) \end{bmatrix} \tag{4.80}$$

were $\Lambda(r,b)$ was previously defined in Equation 4.11. Equation 4.80 encapsulates the kinematics and the closed-loop dynamics of the vehicle in a single model for the entire robotic vehicle.

4.5.2 Pose Estimation Equations

The pose model equations given below are obtained by numerical integration of Equation 4.7 and 4.8 which were presented in Section 4.2. Assuming that the wheels have ground contact with no slippage and no lateral drift, then the pose estimates, (x, y, θ), are obtained as follows [34]:

for $\omega \neq 0$:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \omega T_s \tag{4.81}$$

$$\hat{x}_{k+1} = \hat{x}_k + \frac{v}{\omega}\left(\sin\hat{\theta}_{k+1} - \sin\hat{\theta}_k\right) \tag{4.82}$$

$$\hat{y}_{k+1} = y_k - \frac{v}{\omega}\left(\cos\hat{\theta}_{k+1} - \cos\hat{\theta}_k\right) \tag{4.83}$$

if $\omega = 0$, the pose estimates are obtained with:

$$\hat{\theta}_{k+1} = \hat{\theta}_k \tag{4.84}$$

$$\hat{x}_{k+1} = \hat{x}_k + v\cos\theta_k T_s \tag{4.85}$$

$$\hat{y}_{k+1} = \hat{x}_k + v\sin\theta_k T_s \tag{4.86}$$

where the driving and steering velocities v and ω are assumed to be sampled at the time $t = kT_s$. These velocities can either be simulated from the vehicle model, Equation 4.80, or be actual wheel speed measurements collected with the wheel-drive velocity sensors.

Alternatively, when odometry data from the optical encoder is available, the pose estimates can be obtained using the equations given below [4]:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + \Delta\hat{\theta}_k \tag{4.87}$$

$$\hat{x}_k = \hat{x}_{k-1} + \Delta U_k \cos\hat{\theta}_k \tag{4.88}$$

$$\hat{y}_k = \hat{y}_{k-1} + \Delta U_k \sin\hat{\theta}_k \tag{4.89}$$

where the incremental traveled distance is given by:

$$\Delta u_k = C_m N_k \tag{4.90}$$

with N_k being the pulse count of one sampling interval k, and C_m a conversion constant for converting pulse counts into linear displacement. An expression for C_m is given in Equation 4.40.

The incremental linear displacement of the vehicle geometric center, C_o, is given by:

$$\Delta U_k = \frac{\Delta u_{l,k} + \Delta u_{r,k}}{2} \tag{4.91}$$

and the incremental change in heading angle is calculated with:

$$\Delta\hat{\theta}_k = \frac{\Delta u_{r,k} - \Delta u_{l,k}}{2b} \tag{4.92}$$

where b is half the wheelbase dimension.

Equations 4.81 to 4.92 are used extensively in Section 7.3 when evaluating the localization accuracy and the pose tracking performances of the navigational system.

Chapter 5

Camera Calibration

"If you have a vision, do something with it."

– Anthony J. D'Angelo

Chapter 5

Camera Calibration

5.1 Introduction

A camera calibration is necessary when metric information about a captured scene is required. A calibrated camera allows the measurement of distances under projection on the image plane [35]. Mounted on a mobile robot, it may be used to determine its position and orientation with respect to some known features in the captured scene [36]. This chapter gives the detail of the calibration procedure.

First, Sections 5.2 and 5.4 begin with a review of image formation and ray optics. The more common types of lens aberration such as radial and tangential distortions are discussed. The theory presented in these sections is used later to validate the camera calibration results. Section 5.5 gives a description of the imaging system selected to equip the robotic vehicle. Section 5.6 follows with the mathematical formulation of the camera model. A generic method for solving the unknown camera parameters is presented. Finally, Section 5.7 proceeds with a standard offline calibration procedure and the identification of the camera parameters. A discussion of the results follows.

5.2 Pinhole Camera Model

In the idealistic case, a camera can be modeled as a darkened box provided with a small aperture at one end and an image plane at the other. A fraction of the light rays reflected from an object enters the camera body via the aperture forming a spatial arrangement of light intensities. This process is rectilinear and can be mathematically modeled as a perspective projection.

Using Cartesian coordinates, a perspective projection mapping is given as follows:

$$x_i = f\frac{X}{Z} \tag{5.1}$$

and

$$y_i = f\frac{Y}{Z} \tag{5.2}$$

where f is the focal distance. x_i and y_i are respectively the horizontal and vertical image coordinates of a scene object located at (X, Y, Z). Equations 5.1 and 5.2 together form the so-called pinhole camera model.

A pinhole camera has the drawback of allowing too little light to pass through the aperture to be practical. Enlarging the aperture would allow more light to pass through; however the image would become blurred as light from different sources overlaps [37]. On the other hand, reducing the aperture diameter causes the image formed to be sharper up to a point where light diffraction becomes significant [37]. To solve the aperture size problem, lenses are introduced. A camera equipped with a lens rather than a pinhole permits more light originating from an object to be collected. The lens collects parallel rays of light and redirects them until they cross with the

principal ray[1] at a point called the focus. Convergence of the rays into a single point is possible due to the specific curvature of the lens optics.

Similar to the case of a pinhole camera, the image formed using an ideal lens obeys the same linear relationship given in Equations 5.1 and 5.2. As mentioned above they differ in one aspect, however: brightness. The level of brightness depends on the lens focal ratio. The focal ratio is obtained by dividing the lens focal length by its aperture diameter as follows:

$$N = \frac{f}{D} \tag{5.3}$$

The smaller the focal ratio the more light is concentrated onto the image plane [37].

5.3 Lens Distortion Model

As it is often the case, ideal lenses do not exist. All lenses alter the formation of an image up to a certain degree. The most common types of aberration are the radial and tangential distortion. Lenses also introduce other forms of aberrations such as coma, astigmatism, spherical and chromatic aberration. With current lens manufacturing technologies, these effects are generally negligible when compared to the effects of distortion and will not be covered in the following discussion. The interested reader is referred to [38].

5.3.1 Radial Distortion

Radial lens distortion is an alteration of an image's true scale. While a lens having a large curvature or a small focal ratio helps focus more light onto the

[1] The principal ray is the ray that passes through the center of the lense.

image plane, it has the drawback of warping the projection of straight lines. This has the effect of changing the radial magnification of the image with respect to the optical axis of the camera. This change in magnification is directly linked to the angle of incidence of light rays onto the lens, and in effect decreases the magnification as the radial distance from the optical axis increases. The effect is known as barrel distortion. Modern digital imagers, equipped with lenses of small focal ratios, are often reported to suffer from this type of distortion. The radial lens distortion is often modeled as a power series centered at the principal point with the following form [22]:

$$\begin{bmatrix} \delta x_i^r \\ \delta y_i^r \end{bmatrix} = \begin{bmatrix} x_i\left(k_1 r^2 + k_2 r^4 + \ldots\right) \\ y_i\left(k_1 r^2 + k_2 r^4 + \ldots\right) \end{bmatrix} \tag{5.4}$$

where k_1, k_2, ... are coefficients for radial distortion, x_i and y_i the image coordinates, and $r = \sqrt{x_i^2 + y_i^2}$. Typically, on or two coefficients are sufficient to compensate for the distortion [22].

5.3.2 Tangential Distortion

Tangential distortion refers to a displacement of image points in the horizontal or vertical direction or both at the same time. This form of distortion is caused by misalignment, (i.e., decentration), between the lens's axes [39]. Simple lenses have two axes of symmetry: the optical axis is the axis that joins the center of curvature of the two lens optical surfaces, and the mechanical axis is located at the center of the circular edge of the lens and is determined by the centerline of the machine tool used to grind the lens's optical surfaces [40]. Decentration in the lens produces tangential lens distortion and adds asymmetric radial lens distortion [40]. The expression for tangential distortion is often written in the following form [22]:

$$\begin{bmatrix} \delta x_i' \\ \delta y_i' \end{bmatrix} = \begin{bmatrix} 2p_1 x_i y_i + p_2\left(r^2 + 2x_i^2\right) \\ p_1\left(r^2 + 2y_i^2\right) + 2p_2 x_i y_i \end{bmatrix} \tag{5.5}$$

where p_1, p_2, ... are coefficients for tangential distortion, and r, x_i and y_i are as previously defined. As with the radial distortion model, tangential distortion is symmetric about the principal point.

5.4 Optics

A lens is required to provide the imager with a precise representation of the scene to be captured. The combination of lens/imager determines how much of the scene will be captured, to what level of detail, and how sharp the image will appear. Choosing the right lens for electronic imaging depends on a number of factors. The most critical ones are: the field of view, angular resolution, and depth of field.

5.4.1 Field of View

The field of view determines how much of the scene area is visible through the camera lens. The field of view is affected by the lens focal length, f, and the size of the imager array, d. These parameters are related to the field of view as follows:

$$\theta_{fov} = 2\tan^{-1}\left(\frac{d}{2f}\right) \tag{5.6}$$

There are different ways of specifying the field of view. It can be measured horizontally, vertically, or diagonally with respect to the imager dimensions. A short focal length lens usually yields a large field of view but at the expense of magnifying the perspective effect.

5.4.2 Angular Resolution

Angular resolution is to a digital camera what visual acuity is to the human eye. In an object space, angular resolution is defined as the smallest distance between two scene points that can still be distinguished in a digital image. In image space, it may be thought of as the smallest discrete portion of the field of view pertaining to a single pixel. Neglecting the diffraction effects introduced by the lens, the angular resolution is dependent on two physical factors, the focal length and the imager pixel size. A good approximation of the angular resolution for a particular lens/imager combination can be obtained using the field of view equation:

$$\Delta\theta_{fov} = 2\tan^{-1}\left(\frac{d}{2f}\right) \tag{5.7}$$

where d is substituted by the pixel width. A pixel being the finest detail captured in an image, the angle found corresponds to the smallest angle of incidence for which an object can be distinguished. For rectangular pixels the horizontal and vertical angular resolution differ.

Only for the case where the object is coincident with the optical axis does Equation 5.7 provide an exact value for the angular pixel resolution. For an object imaged in the periphery of the sensor array, it can be shown that the angular pixel resolution monotically decreases with increasing radial distance from the principal point. The proof of this last assertion is outside the scope of the current work and may be of interest only to applications requiring submillimeter sensing accuracy.

5.4.3 Depth of field

The depth of field is defined as the range of distances in which the subject appears sharp in the captured frame. With reference to Figure 5.1, the depth of field is limited by the boundaries defined by D_N and D_F which are respectively the near and far limits of the depth of field. All objects lying within these two limits are considered to be equally sharp. The near and far limits vary with the subject-to-camera distance, S, at which the lens is focused at.

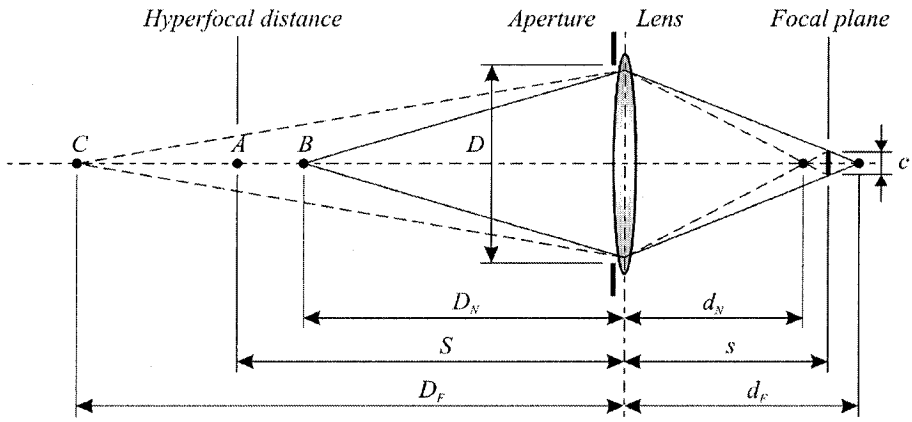


Figure 5.1: Depth of field versus depth of focus.

Assuming a simple thin lens with equal convex curvatures, the near and far limit of depth of field equations can be derived from the thin lens formula [41]:

$$D_N = \frac{Sf^2}{f^2 + cN(S-f)} \tag{5.8}$$

$$D_F = \frac{Sf^2}{f^2 - cN(S-f)} \tag{5.9}$$

where S is the subject-to-camera distance, c the circle of confusion, $N = f / D$ the focal ratio, D the aperture diameter, and f the focal length

In idealized ray optics, the circle of confusion determines the smallest diameter that retains sharp details visible to the human eye. Thus, any object situated outside the depth of field limits defined by Equation 5.8 and 5.9 will appear blurred on the focal plane. Because of the limited resolving power of digital imagers, the circle of confusion cannot be smaller than the physical size of a single pixel.

Focusing the lens at a subject-to-camera distance, S, equal to the hyperfocal distance yield the maximum depth of field. The image of all subjects between B and C will appear equally in focus given a constant circle of confusion. The hyperfocal distance can be calculated from the following equation [42]:

$$H = \frac{f^2}{cN}\left(1+\frac{c}{D}\right) \tag{5.10}$$

For an object A located at the hyperfocal distance and the lens focused on it, it can be shown that Equation 5.9 yields a value of infinity and Equation 5.8 yields a value of approximately $0.5H$. Thus, focusing at a subject distance farther than the hyperfocal distance does not add sharpness to the image, the far limit of the depth of field being already at infinity. Rather, it has the effect of subtracting depth of field from the near limit just in front of the camera.

5.4.4 Depth of Focus

Closely related to the depth of field is the depth of focus. Assuming the lens is focused at the hyperfocal distance, the near limit of the depth of focus, d_N, corresponds to the distance behind the lens where the image of an object C at the far limit of the depth of field will be to a perfect focus. Similarly, the far limit of depth of focus, d_F, corresponds to the distance behind the lens where the image of an object B at the near limit of the depth

of field will be to a perfect focus. The near and far limits of the depth of focus are obtained respectively as follows [41]:

$$d_N = \frac{Sf^2}{(cN+f)(S-f)} \tag{5.11}$$

$$d_F = \frac{Sf^2}{(f-cN)(S-f)} \tag{5.12}$$

With reference to Figure 5.1, for both objects, B and C, to be projected with an equal circle of confusion, c, the focal plane must be located at a distance s from the lens. This distance is the focal plane distance and is given by:

$$s = \frac{Sf}{S-f} \tag{5.13}$$

The depth of focus Equations 5.11 and 5.12 show that in general the distance from the lens to the focal plane is not equal to the magnitude of the focal length, a physical property of the lens. The two can only be made equal when the lens is focused at infinity, and in such case, the near depth of focus becomes coincident with the focal plane.

The usefulness of this subtle distinction between the focal length and the actual distance from the lens to the focal plane will be made obvious in Section 5.7 which covers the camera calibration.

5.5 Camera Selection

The imaging system mounted on the mobile robot serves as a navigational aid. It must be capable of capturing real-time sequences of images while the vehicle is moving in various environments. Yet the images captured must be

of good quality as the visual information contained within is subsequently used to infer its position and orientation with respect to the world. The selection of the proper imaging system components for this particular task, is an important issue. Technically, the imaging system must be:

- capable of capturing undistorted images of moving scenes;
- offer a good compromise between large field of view and high; angular resolution;
- minimize motion blur; and
- minimize blooming.

5.5.1 Imaging System

The low cost consumer-grade electronics camera that satisfies the performance criteria identified above consists of a 1/4 diagonal *Sony ICX098AK* interline transfer CCD array. The array has dimensions of 640×480 pixels^2 with a square pixel size of 5.6×5.6 μm^2. The array is overlaid with a Bayer mosaic filter for color imaging.

The sensor is complemented with a 4.5 mm lens that has a focal ratio of $f/2.2$. The lens is mounted in an M12 $\times$ 0.5 threaded lens holder designed to keep the lens axis perpendicular to the imager plane. Using Equation 5.6, the lens field of view is computed to be 53° with respect to the imager diagonal. As a comparison, the human eye has a field of view of about 90° per eye globe [43]. The imaging system has an angular resolution of 0.0012 rad (0.071°) in both the vertical and horizontal directions. The distortion characteristics of the lens are unknown at this point but will be quantified during the calibration reported in Section 5.7.

The sensor also features a variable speed electronic shutter, capable of delivering up to 30 frames per second at a display resolution of

320×240 pixels^2. In addition, the sensor driver can provide automatic adjustment of the exposure time and a variable image gain. The sensor is reported to have excellent antiblooming characteristics as per the manufacturer [44].

5.6 Perspective Projection

The perspective projection performed by a camera is a special case of projective transformation. Perspective projection is the geometric process by which an ideal camera maps three-dimensional objects onto the two-dimensional surface of its imager. The geometrical transformation is depicted in Figure 5.2 below.

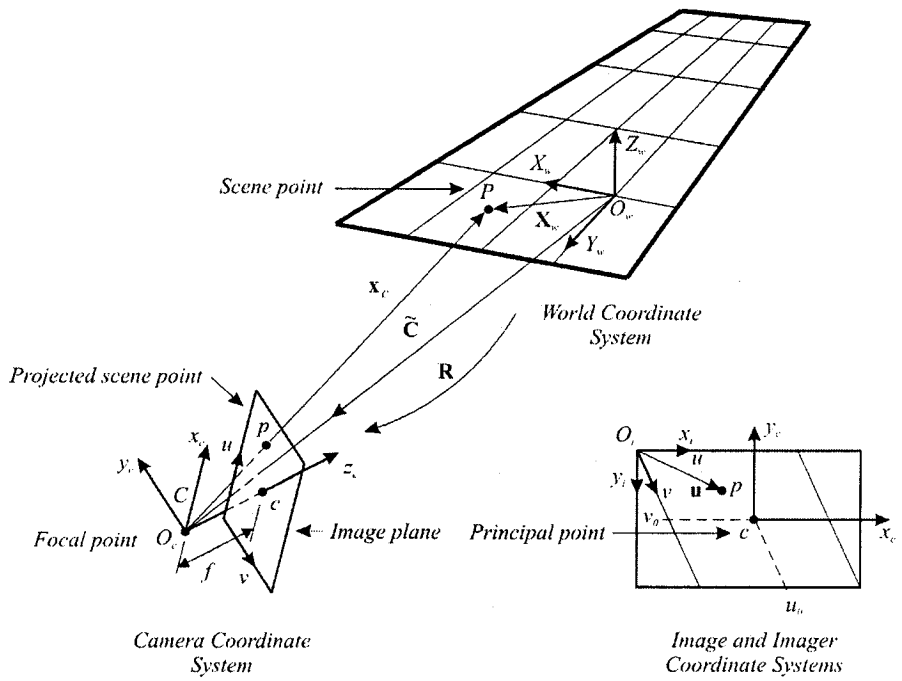


Figure 5.2: Perspective projection.

Four coordinate systems are involved in the transformation. For the sake of clarity, they are described before proceeding.

1. **World Coordinate System** — The World Coordinate System has its origin at the point O_w and coordinate axes (X_w, Y_w, Z_w). The world coordinates define a three-dimensional Euclidean coordinate system.

2. **Camera Coordinate System** — The Camera Coordinate System has its origin at the point O_c and has coordinate axes (x_c, y_c, z_c). The camera coordinates define a three-dimensional Euclidean coordinate system.

3. **Image Coordinate System** — The Image Coordinate System has its origin at the point O_i and has coordinate axes (x_i, y_i). The image coordinates define a two-dimensional Euclidean coordinate system.

4. **Imager Coordinate System** — The Imager Coordinate System has its origin at point O_s, coincident with the image coordinate system origin O_i, and has non-orthogonal coordinate axes (u, v). The imager coordinates define a two-dimensional affine coordinate system.

The projection process is performed via an optical ray reflected from a scene point P that passes through the focal point c and projects at a point p, onto the imager plane. The camera is assumed to have a thin lens with virtually no distortion. The lens is mounted perpendicular to the optical axis and positioned at the focal point c. It is assumed that the image plane is placed between the focal point of the camera and the scene points such that the image inversion that generally occurs is ignored. The principal point c of coordinates (u_0, v_0) is expressed in the imager plane coordinate system, and corresponds to the intersection of the optical axis with the imager plane.

For generality, the image and imager coordinate systems have been kept distinct. The imager coordinate system is an affine transformation of the image coordinate system with respect to shearing and non-uniform scaling,

both being dependent on the imager geometry. The former occurs when the imager has non-square pixels while the latter occurs when the imager array is non-rectangular as a whole. As it is often the case, however, the shearing parameter is neglected [22]. This is true for CMOS and CCD imagers, most of which have rectangular pixel arrays [35].

5.6.1 Intrinsic Parameters

The intrinsic parameters take into account the internal characteristic of an ideal camera. The model derived takes into account the characteristics of the CCD imager and of the lens selected in Section 5.5. Referring to the pinhole camera model, Equations 5.1 and 5.2, the relation that exists between the two-dimensional imager coordinates and the camera coordinates are given by:

$$u = u_0 + k_u f \frac{x_c}{z_c} \tag{5.14}$$

$$v = v_0 - k_v f \frac{y_c}{z_c} \tag{5.15}$$

where k_u and k_v are length-to-pixel conversion factors specific to the CCD imager and have units of pixel/m. For the case of an imager array having a square pixel geometry both conversion factors have the same value. For convenience, the image coordinate origin has been chosen at the upper left corner with the *u-axis* pointing in the same direction as the camera x_c-*axis* and the *v-axis* pointing downward, opposite of the camera y_c-*axis*.

Equations 5.14 and 5.15 are nonlinear in Cartesian space. They are rendered linear with the aid of homogeneous coordinates as follows:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & -\alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{5.16}$$

where α_u and α_v are scaling factors expressed in pixels given by $\alpha_u = k_u f$ and $\alpha_v = k_v f$. w is a scale factor introduced in homogeneous coordinates.

For the case where, $\alpha_u \neq \alpha_v$, non-uniform scaling of the image occurs. The magnitude of non-uniform scaling may also be given in terms of the pixel aspect ratio as follows:

$$\delta = \frac{k_u}{k_v} \tag{5.19}$$

In compact matrix notation, Equation 5.16 becomes:

$$\begin{bmatrix} \mathbf{u}' \\ w \end{bmatrix} = \mathbf{K}\left[\mathbf{I}_{3\times 3} \mid \mathbf{0}\right] \begin{bmatrix} \mathbf{x}_c \\ \mathbf{1} \end{bmatrix} \tag{5.20}$$

where $\mathbf{u}' = (u', v')^\mathsf{T}$ and $\mathbf{x}_c = (x_c, y_c)^\mathsf{T}$. $\mathbf{K}$ is the calibration matrix whose elements are the internal parameters of the camera:

$$\mathbf{K} = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & -\alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

The coefficients of this matrix are the intrinsic parameters of the camera and are expressed in pixels. The parameters are specific to a particular camera and are independent of its position and orientation in space. For generality, the matrix has been expressed including the parameter s which

accounts for the shearing due to non-rectangular pixel. The amount of shearing in degrees of the *v-axis* of the imager with respect to the image y_i-*axis* is calculated as follows:

$$\theta_s = \tan^{-1} s \tag{5.24}$$

The calibration matrix is a special case of an affine transformation. It can be decomposed into independent transformations given that each of the parameters has a geometric interpretation.

5.6.2 Extrinsic Parameters

The extrinsic parameters describe the position and orientation of the camera with respect to a known reference, in this case the world coordinate system. The mapping of the camera coordinate system with respect to the world coordinate system corresponds to a rigid body transformation composed of three rotations and three translations.

With reference to Figure 5.2, a three-dimensional point P represented by the vector $\mathbf{X}_w$ in the world coordinate system is equivalently represented by the vector $\mathbf{x}_c$ in the camera coordinate system, this implies that:

$$\mathbf{x}_c = \mathbf{R}\mathbf{X}_w + \mathbf{t} \tag{5.25}$$

where $\mathbf{t}$ is a 3×1 translation vector from the origin of the camera coordinate frame, O_c, to the origin of the world coordinate frame, O_w, and $\mathbf{R}$ is a 3×3 orthogonal rotation matrix representing the camera orientation with respect to the world coordinate system. Since the world coordinate origin to the camera coordinate origin is given by:

$$\mathbf{R}\tilde{\mathbf{C}} + \mathbf{t} = \mathbf{0} \tag{5.26}$$

where $\widetilde{\mathbf{C}}$ is as translation vector opposite to $\mathbf{t}$. Then, an expression for the translation vector, $\mathbf{t}$, can be recovered as:

$$\mathbf{t} = -\mathbf{R}\widetilde{\mathbf{C}} \tag{5.27}$$

Substituting Equation 5.27 into Equation 5.25 gives:

$$\mathbf{x}_c = \mathbf{R}\left(\mathbf{X}_w - \widetilde{\mathbf{C}}\right) \tag{5.28}$$

The rotation matrix $\mathbf{R}$ expresses three elementary rotations of the camera about the world coordinate axes. Using homogeneous transformations, Equation 5.28 becomes:

$$\begin{bmatrix} \mathbf{x}_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\widetilde{\mathbf{C}} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_w \\ 1 \end{bmatrix} \tag{5.29}$$

$\mathbf{R}$ and $\widetilde{\mathbf{C}}$ contain six extrinsic camera parameters: three rotations and three translations. As opposed to the intrinsic parameters of the camera model, which are constants, the rigid body transformation parameters vary as the pose of the camera changes with respect to the world coordinate frame. The individual rotation need not be known for plane-to-plane homography, only the 3 × 3 matrix of numerical values.

5.6.3 Plane-to-Plane Homography

The complete camera model, including the intrinsic and extrinsic camera parameters, is obtained by substituting Equation 5.29 into 5.20 to give:

$$\begin{bmatrix} \mathbf{u}' \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} \mathbf{X}_w \\ 1 \end{bmatrix} \tag{5.30}$$

where **P** is the 3 × 4 homogeneous perspective projection matrix given by:

$$\mathbf{P} = \mathbf{KR}\left[\mathbf{I}_{3\times3} \mid -\tilde{\mathbf{C}}\right] \tag{5.31}$$

With reference to Figure 5.2, for the case of a plane-to-plane homography, the world coordinate system is chosen such that the plane of scene points has zero Z_w coordinates [35]. This reduces **P** to a 3 × 3 matrix:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{5.32}$$

Since **P** is defined for a given scale factor, w, it has eight degrees of freedom. The matrix is determined uniquely from four correspondences, each giving two constraints as follows:

$$p_{11}x^w + p_{12}y^w + p_{14} - ux^w p_{31} - uy^w p_{32} - up_{34} = 0 \tag{5.33}$$

$$p_{21}x^w + p_{22}y^w + p_{24} - vx^w p_{31} - vy^w p_{32} - vp_{34} = 0 \tag{5.34}$$

where $u = \frac{u'}{w}$ and $v = \frac{v'}{w}$. N such equations can be stacked into a matrix of the form:

$$\begin{bmatrix} x_1^w & y_1^w & 1 & 0 & 0 & 0 & -u_1x_1^w & -u_1y_1^w & -u_1 \\ 0 & 0 & 0 & x_1^w & y_1^w & 1 & -v_1x_1^w & -v_1y_1^w & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i^w & y_i^w & 1 & 0 & 0 & 0 & -u_ix_i^w & -u_iy_i^w & -u_i \\ 0 & 0 & 0 & x_i^w & y_i^w & 1 & -v_ix_i^w & -v_iy_i^w & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N^w & y_N^w & 1 & 0 & 0 & 0 & -u_Nx_N^w & -u_Ny_N^w & -u_N \\ 0 & 0 & 0 & x_N^w & y_N^w & 1 & -v_Nx_N^w & -v_Ny_N^w & -v_N \end{bmatrix} \mathbf{a} = \mathbf{0} \tag{5.37}$$

where

$$\mathbf{a} = \begin{pmatrix} a_{11} & a_{12} & a_{14} & a_{21} & a_{22} & a_{24} & a_{31} & a_{32} & a_{34} \end{pmatrix}^{\mathsf{T}}$$

is the projection matrix **P** arranged as a column vector with the element entries $a_{ij} = p_{ij}$ for $i = 1,2,3$ and $j = 1,2,4$.

Equation 5.37 is the Direct Linear Transformation (DLT) [22]. The superscripts $i = 1,..., N$ denotes the correspondences. The DLT equation has the form $\mathbf{La} = \mathbf{0}$. A method for solving this system is presented next in the context of the calibration procedure.

5.7 Camera Calibration

Camera calibration is the process of determining the intrinsic parameters of a camera. [45]. The procedure consists of solving Equation 5.37 based on a series of plane-to-plane homography between a known calibration target and its image. For Equation 5.37 to be non-singular, eight independent equations are required. These can come from four known non-collinear correspondences, each of these producing two row entries in the DLT matrix. For the case where more than four correspondences are available for homography, Equation 5.37 may be over specified and a solution is obtained using a pseudo-inverse method that minimizes the sum of squares of errors. This is actually recommended to correct for noise in the image measurements [46].

Extracting both the intrinsic and extrinsic parameter via the $a_{11},..., a_{34}$ elements is nontrivial [22]. For that purpose the *Camera Calibration Toolbox* for *Matlab*® was used [23]. The *Toolbox* implements a combination of the pinhole camera and is extended with a lens distortion model. The procedure is divided into three distinct steps: (1) extraction of the feature points, (2) initialization of the extrinsic parameters using a direct linear

transformation (DLT), and (3) refinement of the parameters in the least-square sense.

The first step consists of the extraction of the feature points contained in a calibration target. In this work, the target has dimensions of 270×210 mm^2 and contains a pattern of 9×7 squares arrange as a checkered board of black and white squares. The size of each square is 30×30 mm^2. In total there are 48 corner features on the target. The corners contained in each of the images are extracted to subpixel accuracy using the Harris corner finder [23]. The second step is the initialization of the external parameters **R** and **t** for each of the images contained in the calibration set. A closed-form solution is obtained via a direct linear transformation (DLT) between the known target coordinates $(x_i^w, y_i^w, z_i^w)^\mathsf{T}$ and the corresponding image coordinates of the extracted corner features $(u^i, v^i)^\mathsf{T}$. As a first approximation, the principal point, the skew and the distortion coefficient are set to default values. The focal distances are obtained from the orthogonal vanishing points constraints [23]. The last step involves a refinement of the parameters minimizing the total image reprojection error between the observed image corners of the target $(u^i, v^i)^\mathsf{T}$ and their predicted position $(u_p^i, v_p^i)^\mathsf{T}$ based on the known target features $(x_i^w, y_i^w, z_i^w)^\mathsf{T}$ in the least-square sense using the following objective function [22]:

$$J = \sum_{i=1}^{N} \left(u^i - u_p^i\right)^2 + \sum_{i=1}^{N} \left(v^i - v_p^i\right)^2 \tag{5.39}$$

The optimization is performed on the full set of calibration parameters, including the nine intrinsic subset (α_x, α_y, s, u_0, v_0, k_1, k_2, p_1, p_2), also known as the physical camera parameters [22], and the $6 \times n$ extrinsic subset (R_x, R_y, R_z, t_x, t_y, t_z) where n is the number of images in the calibration set. Estimation of the radial and tangential distortion coefficient (k_1, k_2, p_1, p_2) is optional. The

parameter values obtained during the initialization procedure are used as initial values for the optimization. A refined solution is sought through an iterative gradient descent method until a global minimum is found [23]. Usually, only few iterations are required, however more may be needed if the accuracy of the extracted corners is low.

5.7.1 Procedure

The camera calibrated is the one described in Section 5.5.2. It has a 4.5 mm lens and a focal ratio of $f/2.2$. Prior to the calibration, the camera was focused at the hyperfocal distance. Multiple images of the calibration target were taken by moving the camera around the fixed calibration target with an image resolution of 640×480 pixels^2. In total 16 images were captured. This is more than the number required for determining the unknown parameters. Figure 5.3 and 5.4, respectively shows a mosaic of the calibration image set and the position of the camera at the instant the images were taken.

5.7.2 Results and Discussion

Two calibration trials were performed using the *Camera Calibration Toolbox* and the image set presented in Figure 5.3. The results obtained for the calibration experiments are given in Table 5.1. The parameter values are expressed in pixel units.

The first trial was performed to determine the focal distance and the principal point, all other parameters being known from the manufacturer of the camera. The pixel aspect ratio was set to one and the skew parameter to zero. The distortion coefficients were not estimated. The parameters found in this trial are the ones that are retained for the camera model.

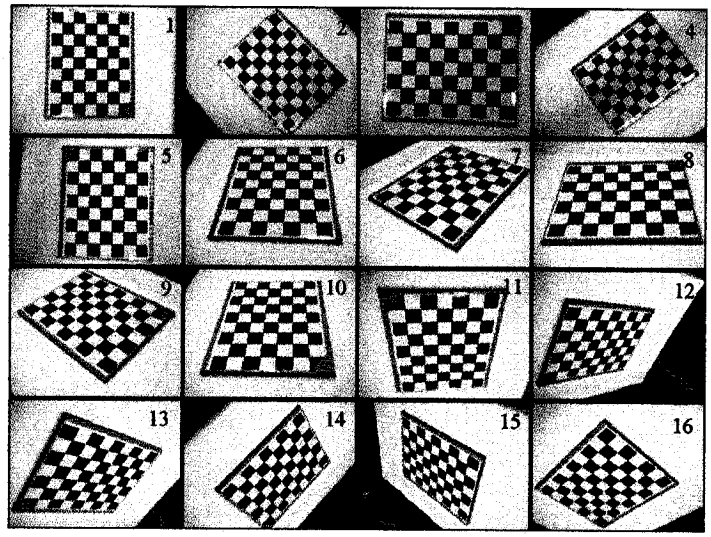


Figure 5.3: Calibration images. The origin of the calibration target is indicated by a dark triangle. The size of each square is 30×30 mm^2.

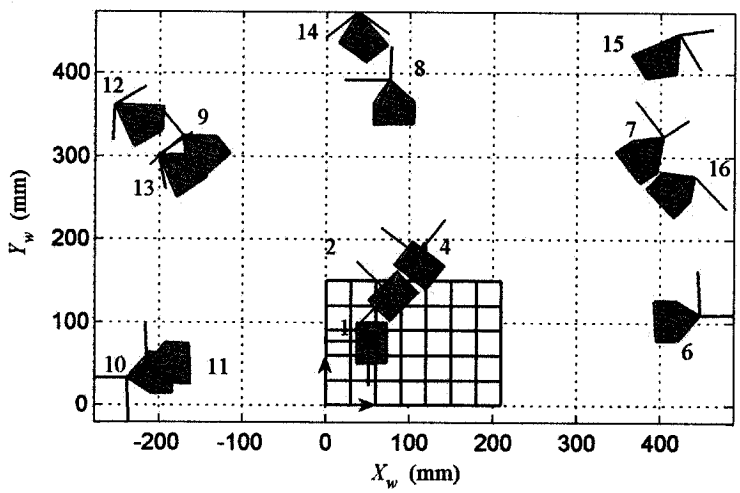


Figure 5.4: Camera pose during calibration. The numbers associated with each of the camera refer respectively to the 16 images of the calibration target (see Figure 5.3).

The second trial was performed to investigate the distortion characteristics of the lens. The distortion coefficients (k_1, k_2, p_1, p_2) were obtained using the radial and tangential distortion model given by Equation 5.4 and 5.5 respectively. All six intrinsic parameters were estimated.

Table 5.1: Calibration results.

Parameters	Trial no. 1[a] (pixels)	Trial no. 2[b] (pixels)
α_x	800.7 ± 4.640	792.5 ± 4.870
α_y	800.7 ± 4.640	796.8 ± 5.218
x_0	362.7 ± 3.100	352.7 ± 9.238
y_0	238.1 ± 4.027	241.8 ± 7.302
s	0 ± 0	-0.0001 ± 0.0018
k_1	0 ± 0	0.1634 ± 0.0456
k_2	0 ± 0	-0.6733 ± 0.3927
p_1	0 ± 0	0.0016 ± 0.0043
p_2	0 ± 0	-0.0010 ± 0.0055
σ_x [c]	0.3859	0.3558
σ_y [c]	0.3407	0.3283

[a] Minimal set of parameter estimated: distortion coefficients and the skew set to 0, aspect ratio set to 1.

[b] All nine parameters estimated using the full set of calibration images.

[c] Standard deviation of the reprojection error expressed in pixels.

Focal Distance — As expected the calibration experiments yielded a focal distance close to the actual lens focal length. The focal distance found is 800.7 pixels ± 4.640 pixels (4.483 mm ± 0.026 mm). The focal distance should be interpreted in the sense of the camera model developed earlier, that is the distance between the lens and the image plane, rather than the actual focal length, a physical property of the lens. Therefore, for an ideal camera and a perfect lens focused at the hyperfocal distance, the theoretical distance

between the focal point and the focal plane can be computed from Equation 5.13. Doing so, the expected focal distance should be equal to 805.8 pixels (4.51 mm).

Principal Point — With reference to Figure 5.5 and 5.6, the center of distortion indicated by the circles coincides with the principal point. As can be seen the principal point departs fairly from the image center. The coordinates found for the principal point are $(u_0, v_0) = (362.7, 238.1)$ compared with $(x_0, y_0) = (320, 240)$ for the image center. The large deviation noticed in the horizontal direction, 42.7 pixels, is believed to be caused by an offset misalignment between the imager center and the optical axis of the lens. In the vertical direction, the discrepancy is small, 1.9 pixels, and is considered acceptable given the uncertainty associated with the parameters.

Distortion — The second trial reported in Table 5.1 shows that lens distortion is insignificant. Radial distortion was estimated using a fourth-order model. The model coefficients found are $k_1 = 0.01634$ and $k_2 = 0.6733$. The tangential distortion coefficients were found to be $p_1 = 0.0016$ and $p_2 = -0.0010$. Figures 5.5 shows a topographical map of the radial component of the lens assembly distortion. The graph indicates that the largest error occurs at the periphery and is equal to 2.5 pixels. Figure 5.6 shows a topographical map of the tangential component of distortion. The graph indicates that the maximum error occurs at the extremities of the sensor diagonal and has a value of 0.8 pixel. The two are relatively small and are neglected in the camera modeling.

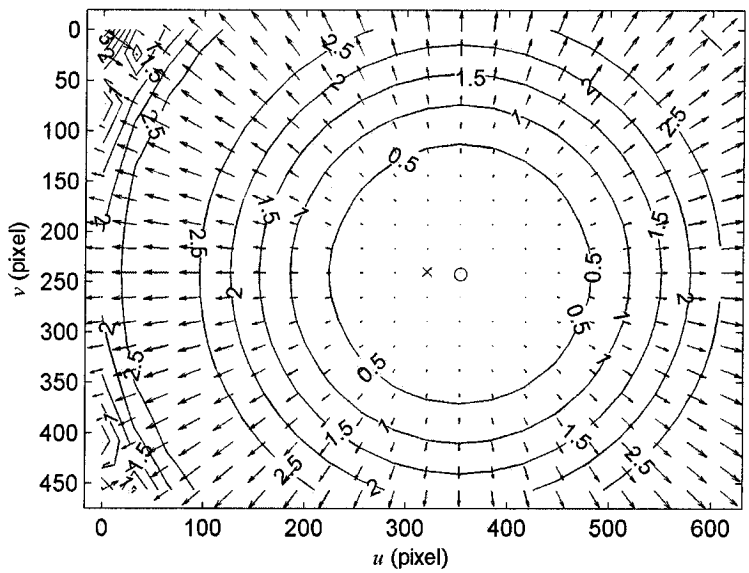


Figure 5.5: Radial component of the lens assembly distortion. Based on a fourth-order model given by Equation 5.4. The distortion coefficients calculated are k_1 = 0.1634, k_2 = -0.6733.

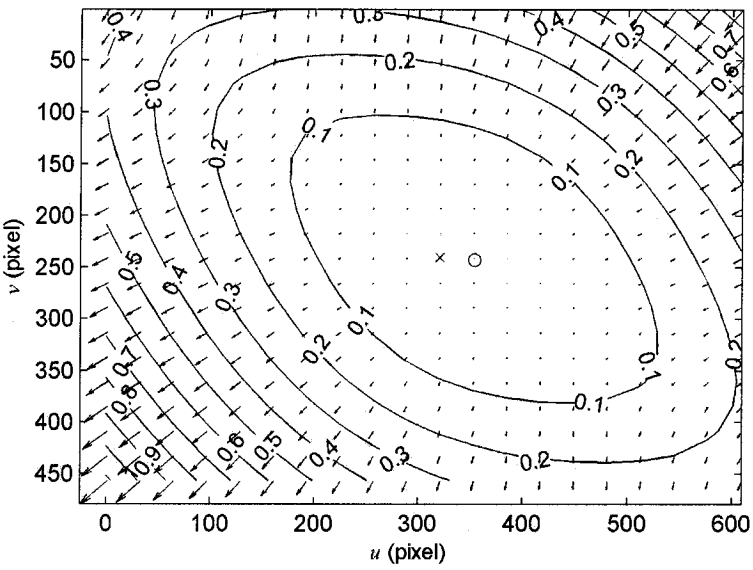


Figure 5.6: Tangential component of the lens assembly distortion. Based on the fourth-order model given by Equation 5.5. The distortion coefficients calculated are p_1 = 0.0016, p_2 = -0.0010.

Parameter Uncertainty — Repeated experiments with different set of images have shown that factors such as camera orientation and exposure time influence the corner detection accuracy. They are believed to be the main cause for the relatively high parameter uncertainty when compared to calibration results reported by others [23], [22]. As reported by Heikkila [22], lower uncertainties are obtained with a set of calibration images captured at angles less than 45° between the camera optical axis and the calibration target surface normal. Image 12 to 16 were both taken at an angle larger than 45°. Large angles alter the resolution at the far end of the target as it appears shortened and makes corner detection in these area less precise. Another source of parameter uncertainty is the sharpness of the images used for the calibration. The sharpness of an image is dependent on the camera focus and the depth of field. Figure 5.7 shows the near and far limits of the depth of field as calculated with Equations 5.8 and 5.9.

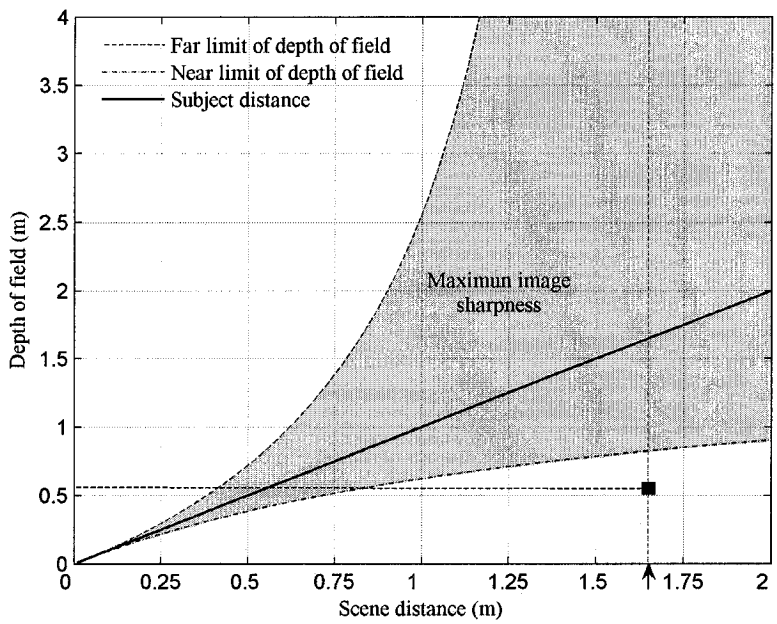


Figure 5.7: Far and near limits of depth of field. The arrow head on the abscissa marks the hyperfocal distance of the camera used in this work. $f = 4.5$ mm, $c = 5.6$ µm.

For the calibration the camera was focused at the hyperfocal distance. Using Equation 5.10, the hyperfocal distance is calculated to be 1.65 m. Focusing the camera at this distance was justified by the fact that it yields the maximum depth of field range in front of the camera. This is a definite requirement for visual navigation as it increases the sharpness of the scene under captured and the accuracy of the pose estimates. During the calibration, however, it was noticed that the image captured were outside this range. In fact, the average camera to calibration target distance was 0.55 m. This is about 2/3 of the near limit of depth of field evaluated to be 0.82 m from Figure 5.7 at a subject distance of 1.65 m.

This had the effect of decreasing the sharpness the corner features, and is a direct consequence of trying to maximize the image area occupied by the calibration target in the image frame. This compromise could have been avoided if a larger target had been used, such that the camera distance to the target is increased to at least the near limit of depth of focus – preferably up to the hyperfocal distance – while keeping the magnification ratio unaltered.

Although Heikkila [22] reports that increasing illumination intensity improves corner detections, this was not noticed when carrying the set of calibration experiments. In fact, reducing the exposure time from 1/15 s to 1/30 s diminished the uncertainty associated with the parameters. Long exposure time caused the target image to saturate in the bright areas and spilled onto the dark region causing a degradation in the ability to detect corners as well as resulting in the formation of glares on the dark regions. It is anticipated that decreasing the camera gain would increase the sharpness of the corner by reducing the amplification of image noise along the edges, but this effect was not evaluated.

Chapter 6

Navigational System

"Equations are just the boring part of mathematics. I attempt to see things in terms of geometry."

– Stephen Hawking

Chapter 6

Navigational System

6.1 Introduction

This chapter looks at how the architecture and models developed in the previous chapters can be integrated into a useful autonomous mobile robot navigation system. The camera is an integral part of the control system.

First, Section 6.2, discusses the problems of absolute and relative localization using visual feedback. It also includes a scope definition of the current work. Section 6.3, presents a geometric model of the robot environment. For that purpose the generic model presented by [47] is complemented. Section 6.4, deals with the rigid body transformations and the notational convention required in Section 6.5. Section 6.5 details a visual pose estimation algorithm which makes use of projective geometry techniques to recover the position and orientation of the robotic vehicle in space. While orientation is determined using a vanishing point, position is recovered using an original approach based on hypothetical homography correspondences. Finally, Section 6.6 concludes the chapter by presenting a simple yet efficient pose tracking controller which uses cubic splines to command the steering velocity.

6.2 Navigational Framework

A conceptual navigational framework is presented below. The framework is based on a hierarchal organization of the environment in which automated navigation tasks are assumed to be performed. For that purpose the environment is divided into multiple local areas. Global localization is achieved via the definition of a topological map wherein local areas are represented as distinct nodes. Since position and orientation are metrically inferred using the camera projection characteristics, a geometric model of the environment is also required.

6.2.1 Localization

As mentioned, absolute and relative localization is achieved via a hierarchal organization of the global and local coordinate frames. At the global level of the hierarchy, a world coordinate system serves as an absolute reference for the local coordinate systems. At the local levels, coordinate systems are assigned to the multiple subareas that divide the environment. Their assignment is illustrated in Figure 6.1.

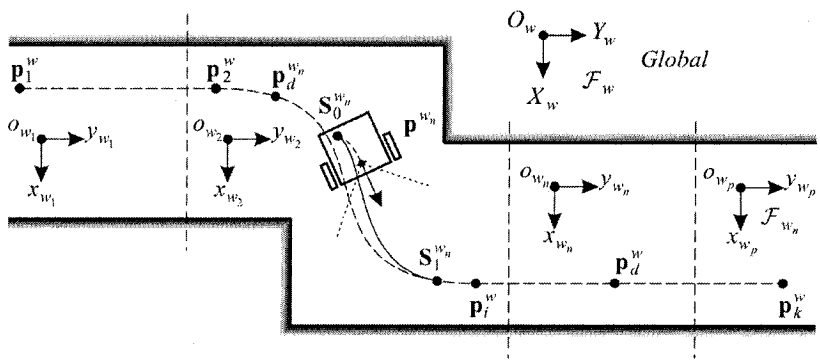


Figure 6.1: A conceptual navigational framework. Local areas are delimited with dashed lines.

The above figure shows a typical environment in which the robot should operate autonomously. For the purpose of vision-based localization, it has been subdivided into multiple local areas delimited with dashed boundaries. Each area is assigned a local reference frame. The reference frame markers can be either natural landmarks or man-made fiducials. Since vision is used as the primary source of sensory perception, the local frames are attributed such that at least one local coordinate frame marker is imaged at any given time.
As will be demonstrated in Section 6.5.3, this constraint can be lifted by making used of hypothetical correspondences. Depending on the navigational strategy adopted, local coordinate frames may have their metric positional information stored in a topological map or be encoded as map nodes[1] recognized by their unique visual signature.

A navigation assignment would take place, starting by positioning the robot into any one of the subareas, say at $\mathbf{p}_2^w$. At this position, the corresponding local frame coordinate is obtained from the visual scenery, that is from a mixed combination of natural landmarks and coordinate frame markers. The robot is then instructed by some external process, an operator for instance, to reach position $\mathbf{p}_k^w$. A planner then determines the sequence of nodes the robot must encounter for reaching the goal. In this pictorial case the robot must reach $\mathbf{p}_i^w$ then $\mathbf{p}_k^w$. Paths are not generated at the planner level. Rather, paths are planned locally using local coordinates. Delegating path planning at a lower level of the hierarchy offers flexibility for replanning locally new paths, especially when unforeseen obstacles are encountered along the route.

---

[1] A node is to a topological map what a frame marker is to a local area. As opposed to frame markers which are visible, nodes are abstract entities of the topological map.

The robot then moves in this frame tracking not only a desired path but also the next coordinate frame marker until positions $\mathbf{p}_i^w$ is reached. Position $\mathbf{p}_i^w$ marks a virtual boundary transition between two adjacent subareas. Upon crossing a virtual boundary, the collected pose measurements are converted to world coordinates, that is to say data from which odometry information can be obtained. Assuming the robot is oriented in such a way that the adjacent local area coordinate frame marker is visible, a transition of coordinate system is performed and navigation resumes in a new local coordinate frame along a newly generated path. This process is repeated until the desired world destination $\mathbf{p}_k^w$ is attained.

The aforementioned procedure puts into perspective the navigation capabilities the robot must be provided with to successfully reach a desired goal. These capabilities are summarized as follows:

- determine the optimal route to follow;
- generate feasible paths within a known environment;
- estimates its position and orientation as it maneuvers;
- control its pose; and
- detect and avoid collision with obstacles.

The set of capabilities actually implemented in the current version of this work is discussed next.

6.2.2 Scope

The current version of this work implements a subset of the navigation framework presented above. The primary objective goal of the mobile robot is to follow a desired path at a travel speed specified by an operator via the human-machine interface. As the robot is first dropped at its initial position it computes its pose, and compares it with the desired configuration. It then sends a first command signal to its actuators and navigation begins. The control problem dealt with is depicted in Figure 6.2.

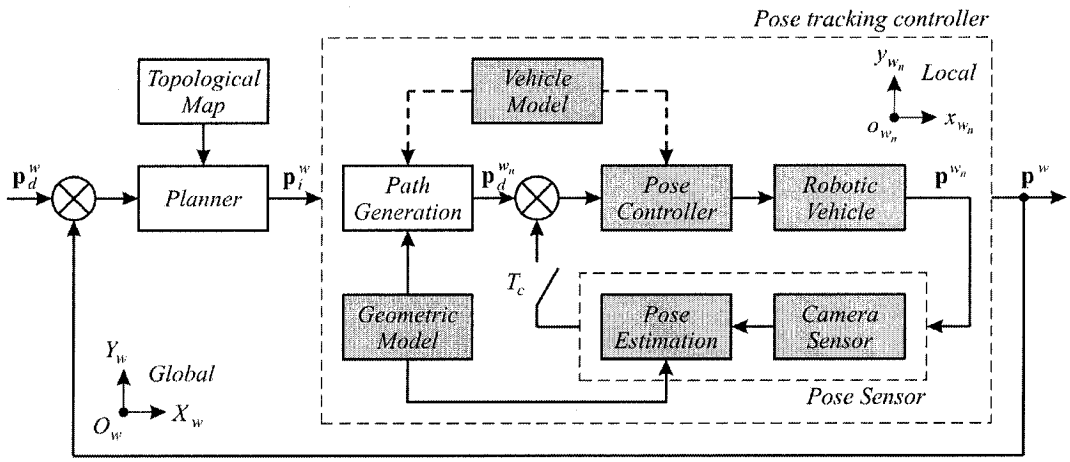


Figure 6.2: Pose tracking controller. Shaded blocks indicate the modules currently implemented. The comparator symbols are used for illustrative purposes only.

The block diagram shows two control layers. The highest layer, the outermost loop, is devoted to the problem of absolute localization and path planning. Planning involves the determination of the sequence of intermediate sub goal positions a robot must encounter before reaching the final destination. Sub goals are generally obtained via some optimization criteria (i.e., shortest path or trajectory smoothness) using a topological map,

a metric map, or a combination of both. The second layer, the innermost loop, takes charge of tracking a desired pose by controlling the motion execution. These two control layers taken together perform the high-level control functions. The low-level control loops in charge of tracking the velocity reference commands on each wheel are not shown here. It can be viewed by substituting the block referenced to as *Robotic Vehicle* with Figure 4.10.

The second control layer structure is the main subject of this chapter. It encapsulates three main sub-components: a geometric model, a visual pose estimation algorithm, and a pose tracking controller. The feedback loop is composed of a camera sensor whose captured images are processed to extract key features in the scene. Using these features, the pose of the vehicle can be resolved provided a known geometric model of the environment is available. As it is suggested in the figure, an internal dynamic model of the vehicle is available to the trajectory controller for verifying the feasability of a proposed path. It may also be used for odometry estimation (refer to Section 4.5.2).

In this work, paths are assumed generated by some other process. Navigation is also limited to local area where change over of local coordinate frames virtually never occurs. In this context, the designation world and local coordinate frame are used interchangeably, and that for the remainder of the present chapter.

6.2.3 Overview of the Control Algorithm

An overview of the control algorithm required to achieve the task described is now presented. It is assumed that the camera internals have been identified during an offline calibration procedure (refer to Section 5.7.2).

1. *Image Acquisition* — An image is captured using the vehicle imaging system.

2. *Line Features* — From the captured image, identify at least two parallel line segments (refer to Section 6.5.1).

3. *Vanishing Point* — From the line segments found, compute the location of the vanishing point corresponding to the dominant scene direction (refer to Section 6.5.2).

4. *Camera Orientation* — From the computed location of the vanishing point, calculate the orientation of the camera with respect to a local coordinate frame (refer to Section 6.5.2).

5. *Camera position* — From planar homography correspondences, compute the position of the camera (refer to Section 6.5.3).

6. *Camera to Local Area Coordinates Transformation* — From the position and orientation of the camera, construct a rigid body transformation $\mathbf{M}_c^w$ (refer to Section 6.4.2).

7. *Vehicle Pose* — From the transformation matrix $\mathbf{M}_r^w$, compute the current pose of the vehicle with respect to the local coordinate frame (refer to Section 6.5.4).

8. *Pose Tracking* — Compare the current to the desired vehicle pose and compute the control action required to correct for heading and lateral position errors (refer to Section 6.6.1).

6.3 Geometric Model

In general, any approach to robot navigation requires a model of the environment in which the robot operates in [21]. A geometric model is especially required when a camera sensor is utilized to infer metrics about position and orientation. [47] suggests that a generic model be defined by a minimal set of three conditions:

1. *it must contain a minimum of two parallel, straight lines that are detectable by the camera sensor; and that,*
2. *the direction of the lines is known with respect to the robot environment; and that,*
3. *the lines are not parallel with the image plane of the camera.*

This model is generic in the sense that it can describe a large class of man-made structure found in an indoor environment [21]. To satisfy the requirements of the task at hand, the generic model is complemented with two additional conditions:

4. *the principal axes of a right-handed local coordinate frame must be positioned according to the following convention:*
   a) *the Y_w-axis is aligned with one of the parallel line segments; pointing toward the dominant vanishing point; and that,*
   b) *the Z_w-axis points upwards perpendicular to the surface containing the lines;*
5. *the distance separating the two parallel lines needs to be known or measurable.*

No assumptions are made about the orientation of the surface containing the lines [21] inasmuch as the position of the local coordinate frame.

6.4 Rigid Body Transformation

Due to the multiplicity of coordinate frames involved, it is deemed appropriate to establish a notational convention for the coordinate transformations before proceeding with the derivation of the pose measurement algorithm. The transformation process between the different coordinate systems is depicted in Figure 6.3.

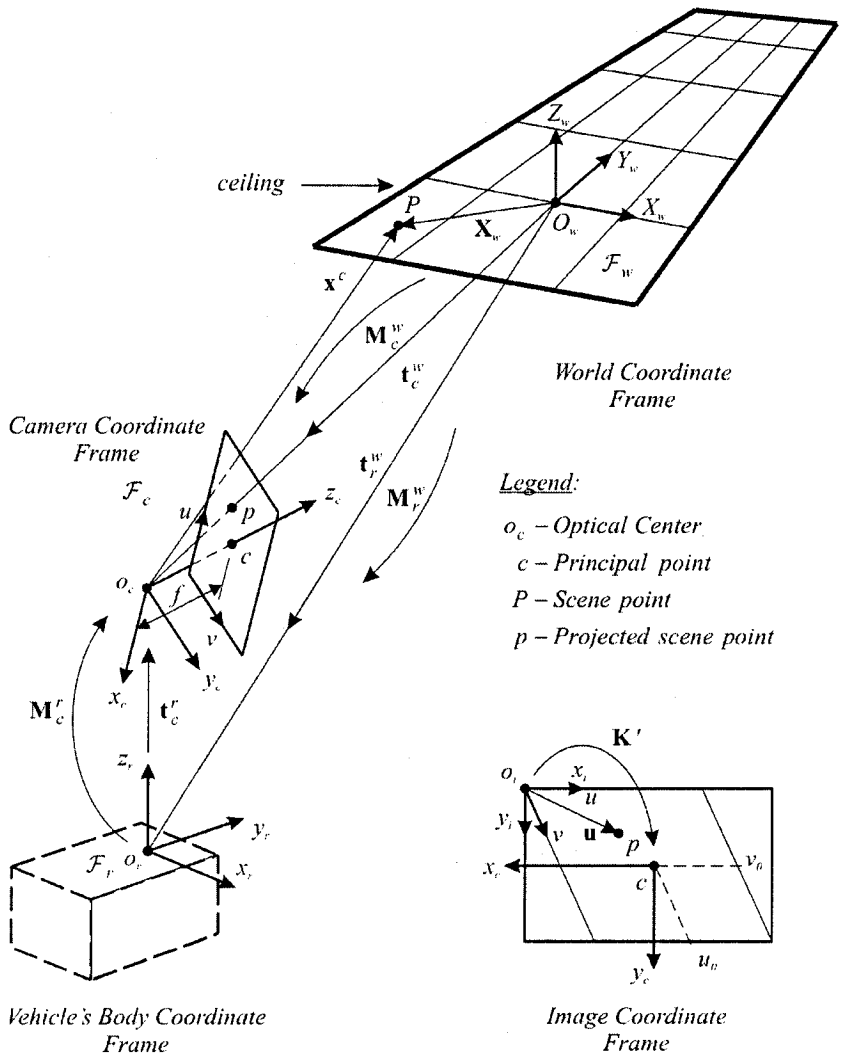


Figure 6.3: Overview of the frame configurations and various coordinate transformations.

6.4.1 Notation

For the purpose of establishing a notational convention, define $\mathbf{M}_a^b$ as an homogeneous transformation between the frame $\mathcal{F}_a$ and the frame $\mathcal{F}_b$. The superscript indicates the reference frame while the subscript indicates the coordinates to be transformed. The transformation $\mathbf{M}_a^b$ is an homogeneous matrix defined as:

$$\mathbf{M}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \mathbf{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix} \tag{6.1}$$

where $\mathbf{t}_a^b = \left(t_{a,x}^b, t_{a,y}^b, t_{a,z}^b\right)^{\mathsf{T}}$ is a translation vector and $\mathbf{R}_a^b$ a rotation matrix expressed with respect to the reference frame, $\mathcal{F}_b$. For the sake of generality, the coordinate transformations presented are assumed having six degrees of freedom; three are from translations and three from rotations. Note that for added clarity, symbol subscripts are appended with the coordinate axis about which the transformation takes place. For an arbitrary angle, φ, the three rotation components are given by:

$$\mathbf{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} \tag{6.2}$$

$$\mathbf{R}_y(\varphi) = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \tag{6.3}$$

$$\mathbf{R}_z(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

where $\mathbf{R}_x(\varphi)$ is a rotation about the *x-axis*, $\mathbf{R}_y(\varphi)$ a rotation about the *y-axis*, and $\mathbf{R}_z(\varphi)$ a rotation about the *z-axis*. The order of composition is important since homogeneous rotations are not commutative. Two different conventions are generally used: the *z-x-z Euler angle* sequence, and the *z-x-y Roll-Pitch-Yaw* convention. Rotations represented using the *Euler angle* sequence are performed about successive moving axes while those represented with the *z-x-y Roll-Pitch-Yaw* convention are performed about fixed axes. The selection of the proper representation is made clear by the context.

6.4.2 Coordinate Transformation

The coordinate frames used in this work are represented in Figure 6.3. The rigid body motion involves three different frames, the world frame, $\mathcal{F}_w$, the vehicle's body frame, $\mathcal{F}_r$, and the camera frame, $\mathcal{F}_c$. Initially, they all have the *z-axis* pointing upward, the *y-axis* pointing forward and the *x-axis* in a direction such as to satisfy the right-hand rule. For generality, the frames are assumed to have six degrees of freedom. The following transformations hold among the coordinate frames:

- *Camera frame* $\mathcal{F}_c$ *to world coordinates frame* $\mathcal{F}_w$.

$$\mathbf{M}_c^w = \begin{bmatrix} \mathbf{R}_c^w & \mathbf{t}_c^w \\ \mathbf{0} & 1 \end{bmatrix} \tag{6.5}$$

where

$$\mathbf{t}_c^w = \left(t_{c,x}^w, t_{c,y}^w, t_{c,z}^w\right)^{\mathsf{T}} \tag{6.6}$$

and

$$\mathbf{R}_c^w = \mathbf{R}_y(\phi_{c,y}^w)\mathbf{R}_x(\psi_{c,x}^w)\mathbf{R}_z(\theta_{c,z}^w) \tag{6.7}$$

- *Camera frame $\mathcal{F}_c$ to vehicle's body coordinates frame $\mathcal{F}_r$.*

$$\mathbf{M}_c^r = \begin{bmatrix} \mathbf{R}_c^r & \mathbf{t}_c^r \\ \mathbf{0} & 1 \end{bmatrix} \tag{6.8}$$

where

$$\mathbf{t}_c^r = \left(t_{c,x}^r, t_{c,y}^r, t_{c,z}^r\right)^{\mathsf{T}} \tag{6.9}$$

and

$$\mathbf{R}_c^r = \mathbf{R}_z(\alpha_{1,z}^r)\mathbf{R}_x(\beta_{2,x}^1)\mathbf{R}_z(\gamma_{c,z}^2) \tag{6.10}$$

- *Vehicle body frame $\mathcal{F}_r$ to world coordinate frame $\mathcal{F}_w$.*

$$\mathbf{M}_r^w = \begin{bmatrix} \mathbf{R}_r^w & \mathbf{t}_r^w \\ \mathbf{0} & 1 \end{bmatrix} \tag{6.11}$$

where

$$\mathbf{t}_r^w = \left(t_{r,x}^w, t_{r,y}^w, t_{r,z}^w\right)^{\mathsf{T}} \tag{6.12}$$

and

$$\mathbf{R}_r^w = \mathbf{R}_y(\phi_{r,y}^w)\mathbf{R}_x(\psi_{r,x}^w)\mathbf{R}_z(\theta_{r,z}^w) \tag{6.13}$$

The first transformation sequence $\mathbf{M}_c^w$ maps the camera coordinate frame, $\mathcal{F}_c$, into the word coordinate frame, $\mathcal{F}_w$. The rotations are combined as per the *y-x-z Roll-Pitch-Yaw* convention. The angles $\left(\phi_{c,y}^w, \psi_{c,x}^w, \theta_{c,z}^w\right)$ are defined as the *roll*, *pitch*, and *yaw* respectively. Postmultiplication is performed using the angle sequence in accordance with the fixed-axes convention [48].

The second transformation sequence $\mathbf{M}_c^r$ maps the camera coordinate frame, $\mathcal{F}_c$, into the vehicle's body coordinate frame, $\mathcal{F}_r$. So as to emulate the functioning of a pan-tilt head mechanism, rotations are represented using the *z-x-z Euler angle* sequence. The angles $\left(\alpha_{1,z}^r, \beta_{2,x}^1, \gamma_{c,z}^2\right)$ are defined as the

pan, *tilt*, and *twist* respectively. The successive rotations about each of the three axes are premultiplied in accordance with the moving-axes convention [48]; the numeral indices indicate the intermediate local frames. For the case where the camera is rigidly attached to the vehicle's body, as is found in the actual implementation, the transformation is assumed to be hand-measured to the exception of the *tilt* angle, $\beta_{2,x}^{1}$, which is kept as an unknown variable.

The third transformation sequence $\mathbf{M}_{r}^{w}$ maps the vehicle body coordinate frame, $\mathcal{F}_r$, into the world coordinate frame, $\mathcal{F}_w$. The rotations are combined as per the *y-x-z Roll-Pitch-Yaw* convention with the angles $\left(\phi_{r,y}^{w}, \psi_{r,x}^{w}, \theta_{r,z}^{w}\right)$ being respectively defined as the *roll*, *pitch*, and *yaw*. In total, the transformation given by Equation 6.11 is characterized by six degrees of freedom, three of which determine the pose of the vehicle. None of the transformation components are directly measurable, however, the transformation can be calculated knowing $\mathbf{M}_{c}^{w}$ and $\mathbf{M}_{r}^{w}$.

Finally, it should be pointed out that the orientation of the camera frame in Figure 6.3 differs from that of Figure 5.2. While the former conforms to a vastly agreed convention in the field of camera calibrations, the latter makes the projection of scene information onto the image plane more natural [49]. This change in orientation of the x_c and y_c camera axes requires the modification of the calibration matrix as follows:

$$\mathbf{K}' = \mathbf{K}\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.14}$$

where $\mathbf{K}$ is the calibration matrix defined in Section 4.4.

It is important to note that the rotation representations given above have singularities. The two compositions of rotation given by Equations 6.7 and 6.13, have singularities at $\psi^{w}_{r,x} = \pm \pi/2$ and $\psi^{w}_{c,x} = \pm \pi/2$, while that of Equation 6.10 is at $\beta^{1}_{2,x} = 0$ or π. At these angles the rotation representations cannot be uniquely determined.

The issue is dealt with globally by reducing the degrees of freedom of the rotation transformations, given by Equations 6.7, 6.10, and 6.13, from three to two. This has the advantage of eliminating the singularities altogether from the representations. Practically, the approach implies locking to fixed values the *twist* angle of the camera and the *roll* angle of the vehicle. While this may seem radical, it is reasonable in the context of indoor mobile robot navigation.

Considering the aforementioned constraints, the limits of the rotation representation are listed in Table 6.1. Note that some of the limits of the rotation representations differ from the intended region of operation. In particular, $\left(\phi^{w}_{c,y}, \psi^{w}_{c,x}, \theta^{w}_{c,z}\right)$ are further restricted by the field of view of the camera.

Table 6.1: Angular limits of the rotation representations.

$\mathcal{F}_c \rightarrow \mathcal{F}_w$	$\mathcal{F}_c \rightarrow \mathcal{F}_r$	$\mathcal{F}_r \rightarrow \mathcal{F}_w$
$-\pi \leq \theta^{w}_{c,z} < \pi$	$-\pi \leq \alpha^{r}_{1,z} < \pi$	$-\pi \leq \theta^{w}_{r,z} < \pi$
$0 \leq \psi^{w}_{c,x} < \pi$	$0 \leq \beta^{1}_{2,x} < \pi$	$-\pi/2 \leq \psi^{w}_{r,x} < \pi/2$
$\phi^{w}_{c,y} = 0$	$\gamma^{2}_{c,z} = 0$	$\phi^{w}_{r,y} = 0$

In situations where the third degree of freedom of rotation might be required, rotation with singularity-free representation such as quaternions could be regarded as an alternative [48].

6.5 Visual Pose Estimation

Pose estimation is a key phase in controlling the trajectory of a robotic vehicle. The pose of the vehicle with respect to its environment can indirectly be found by determining the position and orientation of the camera with respect to the world coordinate frame. Finding the position and orientation of the camera reduces to finding the transformation $\mathbf{M}_c^w$ that maps camera coordinates into the world coordinate frame.

To that effect, an original approach based on the work of Schuster [21] is now presented. Schuster and collaborators demonstrated the usefulness of vanishing points in determining the orientation of a robot. At the same time he pointed out that his method fails to recognize lateral position. Vanishing points are intrinsically not sensitive to translation [35].

The proposed method overcomes this limitation by making use of hypothetical correspondences from which the lateral position of the robot can be measured using projective geometry techniques. The relevance of the method resides in the fact that additional homography features are not required. This is possible by making use of additional information carried by the same line segments that were used for finding the location of the vanishing point on the image plane.

6.5.1 Line Feature Extraction

The current pose of the vehicle can be determined by analyzing key line segments contained in a captured image. The lines are detected using the Standard Hough Transform (SHT). The Hough transform detects and returns a mathematical description of the line segment contained in an image [2]. Before it can be applied, pre-processing of captured images is required.

The image processing is described below. The processing algorithm is a ten-step procedure applied to the real-time image stream captured by the vehicle onboard camera system. It is assumed that the visual scenery of the captured image satisfies the constraints imposed by the geometric model given in Section 6.3. The procedure is as follows:

1. *Image Acquisition* — A 320 × 240 RGB color image is captured with a constant exposure time of 1/30 s and a signal amplification gain set to unity.

2. *Conversion to Grayscale Intensity* — The captured image is converted to a grayscale image using 256 intensity levels (Figure 6.4(a)).

3. *Conversion to Binary Format* — A threshold filter is then applied to the image. Any pixels in the input image with luminance greater than the specified threshold are set to 1 (white) while all other pixels are set to 0 (black). In the example shown in Figure 6.4(b), the threshold is set to 15 % of the maximum intensity level.

4. *Binary Inversion* — The binary image obtained is then inverted such that the image features have a binary value of one (Figure 6.4(b)).

5. *Morphological Thinning* — The image features are morphologically thinned. The operation is performed repeatedly until image objects

shrink to minimally connected strokes preserving the tension and topology of the original features (Figure 6.4(c)).

6. *Parametrization of Line Segments* — From the binary thinned image, line segments are parametrized using the Standard Hough Transform (SHT). The Hough transformation is used to isolate features contained in the image that can be parametrized as line segments, that is $\rho_h = u\cos\theta_h + \sin\theta_h$. The mapping is performed with a default parameter resolution of $\Delta\theta_h = 1°$ and $\Delta\rho_h = 1$ pixel (Figure 6.4(d)).

7. *Extraction of Line Segments* — The Hough space is searched for local maxima from which the line segment parameter pairs, (ρ_h, θ_h), are retrieved (Figure 6.4(e)).

8. *Classification of Line Segments* — The collected line segments are classified as vertical or horizontal. Only the line segments that contribute to a vanishing point along the dominant scene direction are kept, all others are rejected (Figure 6.4(f)).

9. *Computation of Line Intersections* — The intersections among all possible pairs of line segments or with their extensions are computed. For n lines segments identified there are $n(n-1)/2$ intersection possibilities.

10. *Identification of Vanishing Point* — The location of the vanishing point is found at the intersection between the two lines as computed above. For the case where more than two line segments are found, it is computed from a least-square average of the $n(n-1)/2$ set of line intersections.

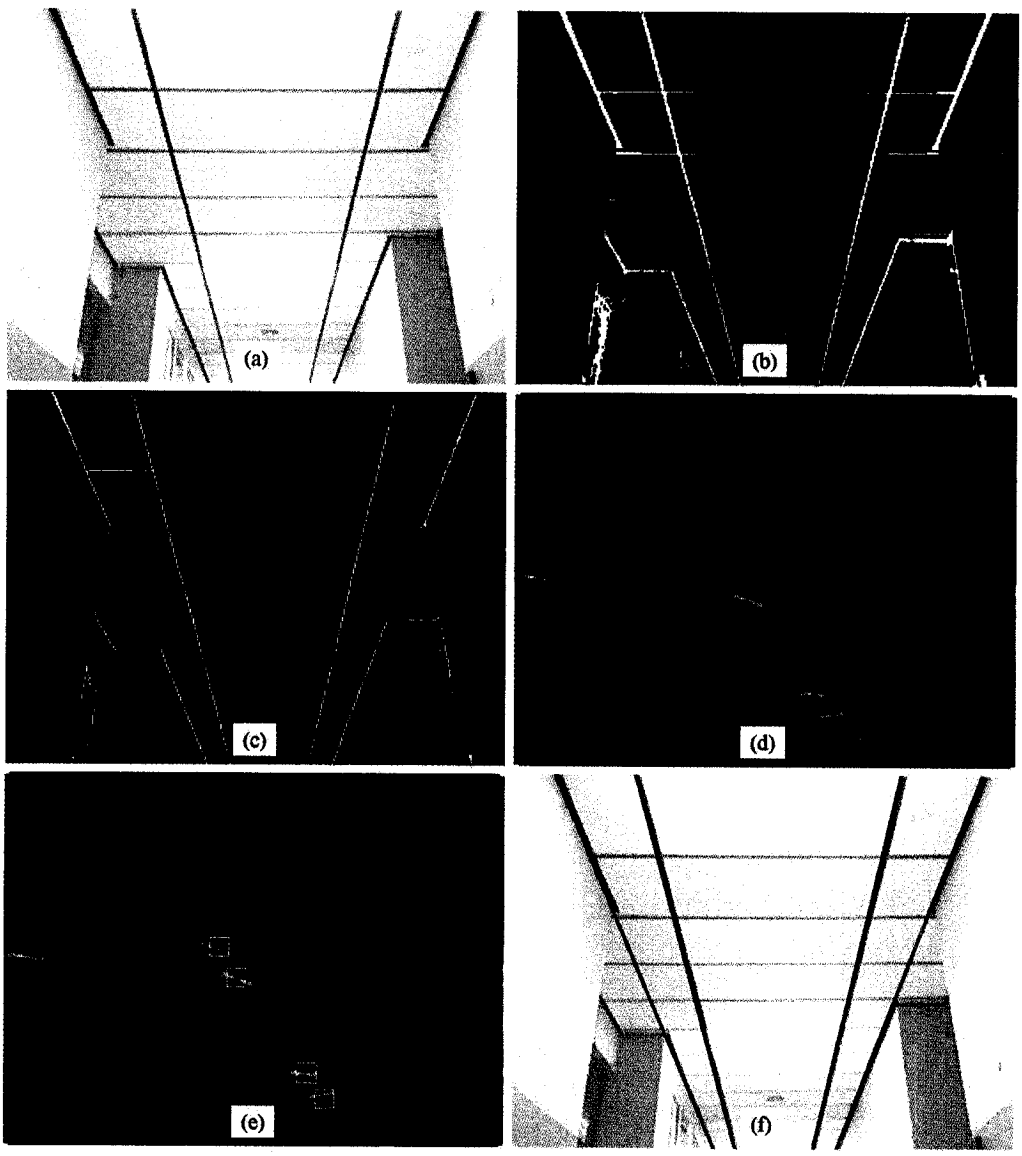


Figure 6.4: Image of a ceiling tile grid structure and dominant line segment extraction. (a) grayscale intensity image of ceiling tile grid, (b) binary image thresholded to 15 %, (c) thinned image, (d) Hough space, (e) Hough space local maximas, and (f) extracted line segments.

6.5.2 Camera Orientation from Vanishing Point

The determination of the camera orientation is inspired by the work of Schuster [21]. Schuster showed that the position and orientation of the camera in space could be decoupled by making use of what is called vanishing points. Vanishing points are image locations where parallel lines seem to converge; a by-product of the camera perspective projection. An image usually contains three vanishing points, one for each of the principal axis. However, one is usually dominant. Two others can be found but are most often computationally ill-conditioned [50].

Vanishing points are points at infinity. In homogeneous coordinates, points at infinity are equivalent to directions [35]. Therefore, for lines parallel to any of the world coordinate frame, $\mathcal{F}_w$, principal axes, their locations can be found from [35]:

$$\mathbf{V} = \mathbf{PD} \tag{6.15}$$

where

$$\mathbf{P} = \mathbf{K}'\left[\mathbf{I}_{3\times 3} \mid \mathbf{0}\right]\left(\mathbf{M}_c^w\right)^{-1} \tag{6.16}$$

is a 4 × 4 homogeneous projection matrix (see Section 5.6.3) and $\mathbf{V}$ a 4 × 3 matrix of column vectors, that is

$$\mathbf{V} = \left[\mathbf{v}_x \;\; \mathbf{v}_y \;\; \mathbf{v}_z\right]$$

represents the image location of the vanishing points corresponding to each of the world coordinate frame principal axis directions. The principal directions are concatenated such that:

$$\mathbf{D} = \left[\mathbf{d}_x \;\; \mathbf{d}_y \;\; \mathbf{d}_z\right]$$

where

$$\mathbf{d}_x = [1 \quad 0 \quad 0 \quad 0]^T, \ \mathbf{d}_y = [0 \quad 1 \quad 0 \quad 0]^T, \text{ and } \mathbf{d}_z = [0 \quad 0 \quad 1 \quad 0]^T$$

Substituting Equation 6.16 into 6.15, it follows that

$$\mathbf{V} = \mathbf{K}'\left[\mathbf{I}_{3\times3} \mid \mathbf{0}\right]\left(\mathbf{M}_c^w\right)^{-1}\mathbf{D} \tag{6.17}$$

Since only the vanishing point formed by lines parallel the world coordinate frame axis direction, $\mathbf{d}_y$, is sought, Equation 6.17 reduces to:

$$\mathbf{v}_y = \mathbf{K}'\left[\mathbf{I}_{3\times3} \mid \mathbf{0}\right]\left(\mathbf{M}_c^w\right)^{-1}\mathbf{d}_y \tag{6.18}$$

Performing the matrices multiplication, and recalling from Section 6.4.2 that, $\phi_{c,y}^w$ was set to zero, it follows that:

$$\mathbf{v}_y = \begin{bmatrix} \alpha_u \sin\theta_{c,z}^w - s\cos\psi_{c,x}^w \cos\theta_{c,z}^w + u_0 \sin\psi_{c,x}^w \cos\theta_{c,z}^w \\ \alpha_v \cos\psi_{c,x}^w \cos\theta_{c,z}^w + v_0 \sin\psi_{c,x}^w \cos\theta_{c,z}^w \\ \sin\psi_{c,x}^w \cos\theta_{c,z}^w \end{bmatrix} \tag{6.19}$$

whose inhomogeneous coordinates are

$$u_v = \alpha_u \frac{\tan\theta_{c,z}^w}{\sin\psi_{c,x}^w} - s\cot\psi_{c,x}^w + u_0 \tag{6.20}$$

$$v_v = \alpha_v \cot\psi_{c,x}^w + v_0 \tag{6.21}$$

where u_v and v_v are the image coordinates of the vanishing point. As Equations 6.20 and 6.21 suggest vanishing points are invariant under translation [35]. This property of vanishing points makes possible the

recovery of the camera orientation irrespectively of its position in space. The location of the vanishing point as a function of the camera orientation given by Equations 6.20 and 6.21, is illustrated in Figure 6.5.

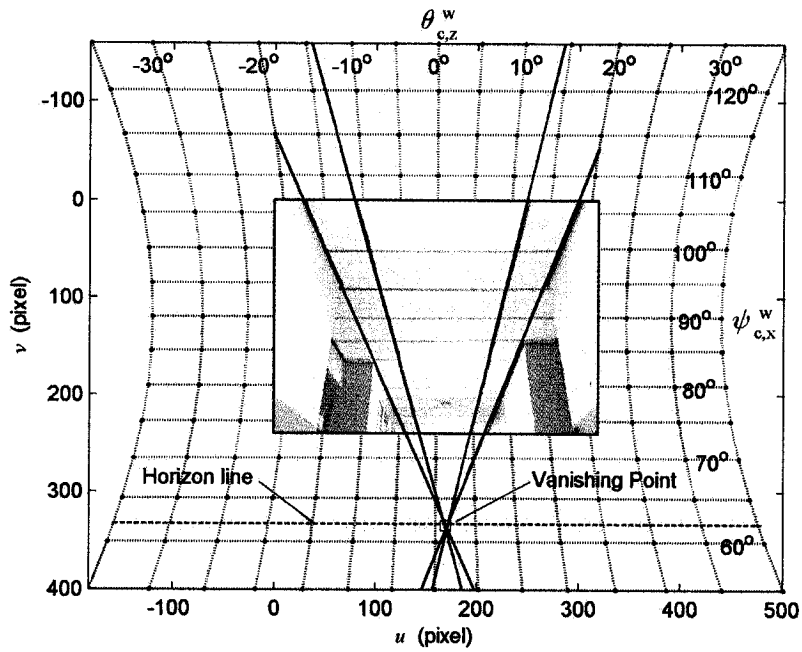


Figure 6.5: Location of the vanishing point as a function of the camera orientation.

Figure 6.5 shows a grid superimposed on a typical image captured during navigation. When $\psi^{w}_{c,x}$ is varied while $\theta^{w}_{c,x}$ is kept fixed the loci of vanishing points forms a curve in the image plane. Similarly, if $\theta^{w}_{c,x}$ is varied while $\psi^{w}_{c,x}$ is kept constant, the loci of vanishing points forms a horizontal line often called the horizon line. Only for the special case where the camera is pointing in a direction parallel to the world coordinate Y_w-*axis*, for which $\psi^{w}_{c,x} = 90°$ and $\theta^{w}_{c,x} = 0°$, is the vanishing point found at the center of the image. For the particular scene shown above, the vanishing point was found at $(u_v, v_v) = (96, 469)$ which corresponds camera orientation angles of $\theta^{w}_{c,x} = -4.1°$ and $\psi^{w}_{c,x} = 66.5°$.

Equations 6.20 and 6.21 predict the location of the vanishing point in image coordinates given a known camera orientation. Conversely, knowing the image coordinates, (u_v, v_v), of the vanishing point, found using the method described in Section 6.5.1, the orientation of the camera with respect to the world coordinate frame, $\mathcal{F}_w$, can be recovered. Thus, from Equation 6.20 and 6.21, the camera *pitch* and *yaw* angles are given respectively by:

$$\psi_{c,x}^{w} = \arctan\left(\frac{\alpha_v}{v_v - v_0}\right) \tag{6.24}$$

$$\theta_{c,z}^{w} = \arctan\left(\frac{\left(u_v - u_0 + s\cot\psi_{c,x}^{w}\right)\sin\psi_{c,x}^{w}}{\alpha_u}\right) \tag{6.25}$$

Since the *pitch* angle $\psi_{c,x}^{w}$ is assumed constant during navigation, Equation 6.24 indicates that the location of the horizon line, in image space, is known. As a result, finding the location of the vanishing point merely becomes finding the intersection of this line with any one line parallel to the Y_w-*axis*. Consequently, the vanishing point can be found even for situations where only one parallel line is detected by the camera.

6.5.3 Camera Position from Planar Homography

As was presented in Section 6.4, the task of finding the position of the camera with respect to the world coordinates is equivalent to that of finding the translation vector $\mathbf{t}_c^w$ of the rigid body transformation $\mathbf{M}_c^w$. This is usually achieved by computing a plane-to-plane homography using a minimum set of two point feature correspondences between the world and image coordinates.

In this work, a slightly different approach is proposed. It uses scene line segments and their projections on the image plane rather than an array of known correspondences. For that purpose, the two line segments that were used in the calculation of the vanishing point are reutilized. The originality of the method resides in the use of two hypothetical scene points, (x_1^w, y_1^w) and (x_2^w, y_2^w), as can be seen in Figure 6.6, whose positions in space are restricted according to the following rules:

1. *First, two hypothetical points are positioned in space, one on each of the two parallel line segments.*

2. *Second, the points are constrained to lie on a line perpendicular to the parallel line segments. The spacing between the two parallel lines is used as a metric.*

3. *Third, the hypothetical points are constrained to move along the parallel line segments as the vehicle moves such that they are positioned at a constant distance ahead of the vehicle at all times.*

4. *Fourth (optional), one of the two hypothetical points is positioned such that it is coincident with the origin of the world coordinate system.*

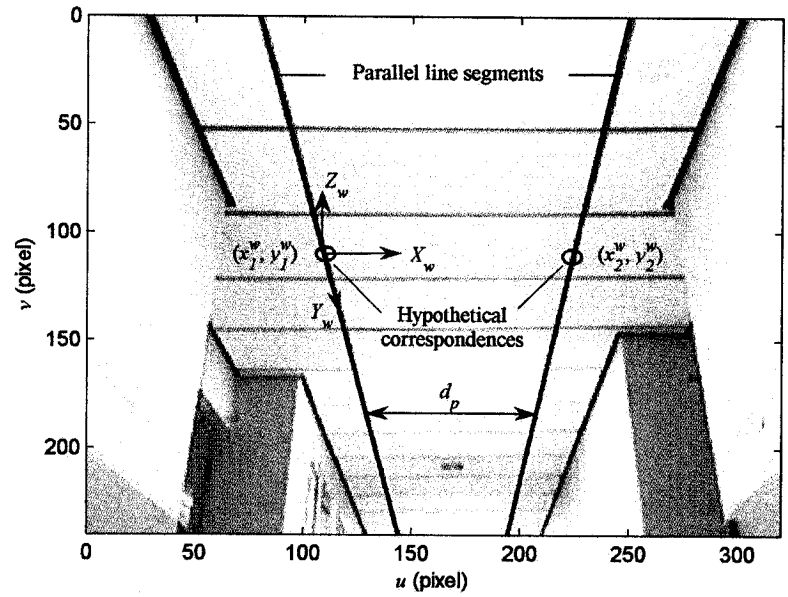


Figure 6.6: Hypothetical correspondences and recovery of the lateral position.

Strictly observing the above rules, a solution for the translation vector is obtained by forming a correspondence for each of the hypothetical points. Using Equations 5.33 and 5.34, each correspondence produces two equations. It then follows that:

$$p_{11}x_1^w + p_{12}y_1^w + p_{14} - u_1x_1^w p_{31} - u_1y_1^w p_{32} - u_1p_{34} = 0 \tag{6.26}$$

$$p_{21}x_1^w + p_{22}y_1^w + p_{24} - v_1x_1^w p_{31} - v_1y_1^w p_{32} - v_1p_{34} = 0 \tag{6.27}$$

$$p_{11}x_2^w + p_{12}y_2^w + p_{14} - u_2x_2^w p_{31} - u_2y_2^w p_{32} - u_2p_{34} = 0 \tag{6.28}$$

$$p_{21}x_2^w + p_{22}y_2^w + p_{24} - v_2x_2^w p_{31} - v_2y_2^w p_{32} - v_2p_{34} = 0 \tag{6.29}$$

where the p_{ij}, for i = 1,2,3 and j = 1,2,4 are the components of the plane-to-plane projection matrix (see Section 5.6.3) with p_{14}, p_{24}, and p_{34}

known up to two of the translation components of $\mathbf{t}_c^w = (t_{c,x}^w, t_{c,y}^w, t_{c,z}^w)^\mathsf{T}$. The translation $t_{c,y}^w$ is specified as a constant look ahead distance in front of the vehicle in compliance with the third rule mentioned above. This set of equations describes a system in eight unknowns: (u_1, v_1), $(t_{c,x}^w, t_{c,z}^w)$, (u_2, v_2), and (x_2^w, y_2^w). Four additional equations are required for a unique solution to be obtained. Two equations arise from the mathematical formulation of the second rule, that is:

$$x_2^w = x_1^w + d_p \tag{6.30}$$

$$y_2^w = y_1^w \tag{6.31}$$

where d_p is the distance between the parallel lines. Two more equations may come from the projection of the parallel line segments into the image plane. As a consequence of the Hough transform, these lines now have a parametrization of the form (see Section 6.5.1):

$$v_1 = m_1 u_1 + b_1 \tag{6.32}$$

$$v_2 = m_2 u_2 + b_2 \tag{6.33}$$

where $m_{1,2}$ and $b_{1,2}$ are respectively the slopes and intercepts of both line segments. These parameters are expressed in pixels.

Equation 6.26 to 6.33 form a set of eight equations in eight unknowns. Because they are nonlinear in the unknowns, a solution is difficult to obtain without using nonlinear optimization methods. Even if a solution is found, sign ambiguities would still exist due to their quadratic nature.

A closed-form solution can be obtained by observing the fourth rule which suggests constraining one of the hypothetical points to coincide with the origin of the world coordinate system. For that purpose, it is convenient to choose x_1^w. As a result, the products with x_1^w, y_1^w, and y_2^w drop out of the equations. The reduced set is given as follows:

$$p_{14} - u_1 p_{34} = 0 \tag{6.34}$$

$$p_{24} - m_1 u_1 p_{34} - b_1 p_{34} = 0 \tag{6.35}$$

$$d_p p_{11} + p_{14} - u_2 d_p p_{31} - u_2 p_{34} = 0 \tag{6.36}$$

$$d_p p_{21} + p_{24} - d_p m_2 u_2 p_{31} - d_p b_2 p_{31} - m_2 u_2 p_{34} - b_2 p_{34} = 0 \tag{6.37}$$

Substitution of Equations 6.30 to 6.33 into Equations 6.26 to 6.29 results in a set of four equations that can be solved simultaneously for the four unknowns $(t_{c,x}^w, t_{c,z}^w)$ and (u_1, u_2).

The solution is obtained in two stages. First, the system of Equations 6.34 to 6.37, is solved for the variables p_{14}, p_{34}, u_1, and u_2. The solution gives:

$$p_{34} = \frac{m_2 p_{24} - m_1 d_p (p_{21} - b_2 p_{31} - m_2 p_{11})}{m_2 b_1 - m_1 b_2} \tag{6.38}$$

$$u_1 = \frac{p_{24} - b_1 p_{34}}{m_1 p_{34}} \tag{6.39}$$

$$p_{14} = u_1 p_{34} \tag{6.40}$$

$$u_2 = \frac{d_p p_{11} + p_{14}}{d_p p_{31} + p_{34}} \tag{6.41}$$

where p_{14}, p_{24}, and p_{34} are unknown up to a translation for which a solution is sought.

Second, the translation is recovered by expanding the last column of the projection matrix such that the components of the translation vector are explicitly revealed. To carry out this process, let:

$$\mathbf{P} = \mathbf{F}\left[\mathbf{I}_{3\times 3} \left| -\mathbf{t}_c^w \right.\right] \tag{6.42}$$

where $\mathbf{F} = -\mathbf{K}'\left(\mathbf{R}_c^w\right)^T$ is a 3×3 matrix. Expanding the projection matrix elements p_{14}, p_{24}, and p_{34} gives:

$$p_{14} = f_{11}t_{c,x}^w + f_{12}t_{c,y}^w + f_{13}t_{c,z}^w \tag{6.43}$$

$$p_{24} = f_{21}t_{c,x}^w + f_{22}t_{c,y}^w + f_{23}t_{c,z}^w \tag{6.44}$$

$$p_{34} = f_{31}t_{c,x}^w + f_{32}t_{c,y}^w + f_{33}t_{c,z}^w \tag{6.45}$$

where the f_{ij}, for $i = 1,2,3$ and $j = 1,2,3$, are the elements of $\mathbf{F}$. Equation 6.38 to 6.41 are rearranged so as to form the following two equalities:

$$p_{34} = Ap_{24} + B \tag{6.46}$$

$$p_{14} = Cp_{24} + D \tag{6.47}$$

where

$$A = \frac{m_2}{m_2 b_1 - m_1 b_2}, \quad B = -\frac{m_1 d_p\left(f_{21} - b_2 f_{31} - m_2 f_{11}\right)}{m_2 b_1 - m_1 b_2}, \quad C = \frac{\left(1 - b_1 A\right)}{m_1}, \quad D = \frac{b_1}{m_1} B.$$

Substituting Equations 6.43 to 6.45 into Equations 6.46 and 6.47 and collecting coefficients, it follows that:

$$(f_{31} - Af_{21})t^w_{c,x} + (f_{32} - Af_{22})t^w_{c,y} + (f_{33} - Af_{23})t^w_{c,z} = B \tag{6.48}$$

$$(f_{11} - Cf_{21})t^w_{c,x} + (f_{12} - Cf_{22})t^w_{c,y} + (f_{13} - Cf_{23})t^w_{c,z} = D \tag{6.49}$$

which can be rearranged in matrix form such that:

$$\mathbf{t}^w_c = \begin{bmatrix} t^w_{c,x} \\ t^w_{c,y} \\ t^w_{c,z} \end{bmatrix} = \begin{bmatrix} (f_{31} - Af_{21}) & 0 & (f_{33} - Af_{23}) \\ 0 & 1 & 0 \\ (f_{11} - Cf_{21}) & 0 & (f_{13} - Af_{23}) \end{bmatrix}^{-1} \begin{bmatrix} B - (f_{32} - Af_{22}) \\ 1 \\ D - (f_{12} - Cf_{22}) \end{bmatrix} t^w_{c,y} \tag{6.50}$$

where $t^w_{c,y}$ is the look ahead distance and is known a priori.

Equation 6.50 gives a closed-form solution for the recovery of the translation vector, $\mathbf{t}^w_c$, from two hypothetical scene points. If desired the projection of these points onto the image plane can be found using the projection matrix given by Equation 6.42.

The advantage of the proposed approach, as demonstrated above, resides in the fact that the locations of the hypothetical points, (x^w_1, y^w_1) and (x^w_2, y^w_2), in the world coordinate frame have no dependancy on the camera orientation; provided that the conditions imposed by the geometric model are first met. As a consequence, the hypothetical correspondences may be located outside the field of view and yet can be projected onto the image plane. However, their projection would lie at the periphery of the active image area.

6.5.4 Recovery of the Vehicle's Pose

The pose of the vehicle with respect to the world coordinate frame, $\mathcal{F}_w$, cannot be recovered directly without first resolving the camera coordinates into the vehicle's coordinate frame, $\mathcal{F}_r$. Using the transformations that were presented in Section 6.4.2, the pose is recovered as follows.

It can be shown that the position and orientation of the vehicle's body with respect to the world coordinate frame, $\mathcal{F}_w$, is given by:

$$\mathbf{M}_r^w = \mathbf{M}_c^w \left(\mathbf{M}_c^r\right)^{-1} \tag{6.51}$$

However, since $\mathbf{M}_c^w$ is only known up to an angle, $\beta_{2,x}^1$, the *pitch* angle of the camera, the unknown transformations must be solved individually. Substituting Equations 6.5, 6.8, and 6.11 into Equation 6.51 and rearranging so as to eliminate the inverse transformation, yields:

$$\begin{bmatrix} \mathbf{R}_c^w & \mathbf{t}_c^w \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_r^w & \mathbf{t}_r^w \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_c^r & \mathbf{t}_c^r \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{6.52}$$

Equating the right and left-hand side of Equation 6.52 element-by-element, the following two equalities result:

$$\mathbf{R}_r^w = \mathbf{R}_c^w \left(\mathbf{R}_c^r\right)^{-1} \tag{6.53}$$

$$\mathbf{t}_r^w = \mathbf{t}_c^w - \mathbf{R}_r^w \mathbf{t}_c^r \tag{6.54}$$

Substituting Equations 6.7, 6.10, and 6.13 into Equation 6.53, it follows that:

$$\mathbf{R}_y(\phi_{r,y}^w)\mathbf{R}_x(\psi_{r,x}^w)\mathbf{R}_z(\theta_{r,z}^w) = \mathbf{R}_c^w \mathbf{R}_z^{\mathsf{T}}(\gamma_{c,z}^2)\mathbf{R}_x^{\mathsf{T}}(\beta_{2,x}^1)\mathbf{R}_z^{\mathsf{T}}(\alpha_{1,y}^r) \tag{6.55}$$

With reference to Table 6.1, ${}^{w}\phi_{r,y}$ and ${}^{2}\gamma_{c,z}$ were set to zero, hence Equation 6.55 reduces to:

$$\mathbf{R}_x({}^{w}\psi_{r,x})\mathbf{R}_z({}^{w}\theta_{r,z}) = \mathbf{R}_c^{w}\mathbf{R}_x^{\mathsf{T}}({}^{1}\beta_{2,x})\mathbf{R}_z^{\mathsf{T}}({}^{r}\alpha_{1,y}) \tag{6.56}$$

This last equation contains three unknowns, the *pitch* angle of the vehicle, ${}^{w}\psi_{r,x}$, the *tilt* angle of the camera, ${}^{1}\beta_{2,x}$, and the *yaw* angle of the vehicle, ${}^{w}\theta_{r,z}$.

Since rotation transformations are nonlinear, a solution cannot be explicitly obtained. Rather, the matrices must be decomposed element-by-element which yields nine equations in three unknowns as follows:

$$r_{11}\cos{}^{r}\alpha_{1,z} + (r_{12}\cos{}^{1}\beta_{2,x} - r_{13}\sin{}^{1}\beta_{2,x})\sin{}^{r}\alpha_{1,z} = \cos{}^{w}\theta_{r,z} \tag{6.57}$$

$$r_{11}\sin{}^{r}\alpha_{1,z} - (r_{12}\cos{}^{1}\beta_{2,x} - r_{13}\sin{}^{1}\beta_{2,x})\cos{}^{r}\alpha_{1,z} = -\sin{}^{w}\theta_{r,z} \tag{6.58}$$

$$r_{12}\sin{}^{1}\beta_{2,x} + r_{13}\cos{}^{1}\beta_{2,x} = 0 \tag{6.59}$$

$$r_{21}\cos{}^{r}\alpha_{1,z} + \left(r_{22}\cos{}^{1}\beta_{2,x} - r_{23}\sin{}^{1}\beta_{2,x}\right)\sin{}^{r}\alpha_{1,z} = \cos{}^{w}\psi_{r,x}\sin{}^{w}\theta_{r,z} \tag{6.60}$$

$$r_{21}\sin{}^{r}\alpha_{1,z} - \left(r_{22}\cos{}^{1}\beta_{2,x} - r_{23}\sin{}^{1}\beta_{2,x}\right)\cos{}^{r}\alpha_{1,z} = \cos{}^{w}\psi_{r,x}\cos{}^{w}\theta_{r,z} \tag{6.61}$$

$$r_{22}\sin{}^{1}\beta_{2,x} + r_{23}\cos{}^{1}\beta_{2,x} = -\sin{}^{w}\psi_{r,x} \tag{6.62}$$

$$r_{31}\cos{}^{r}\alpha_{1,z} + \left(r_{32}\cos{}^{1}\beta_{2,x} - r_{33}\sin{}^{1}\beta_{2,x}\right)\sin{}^{r}\alpha_{1,z} = \sin{}^{w}\psi_{r,x}\sin{}^{w}\theta_{r,z} \tag{6.63}$$

$$r_{31}\sin{}^{r}\alpha_{1,z} - \left(r_{32}\cos{}^{1}\beta_{2,x} - r_{33}\sin{}^{1}\beta_{2,x}\right)\cos{}^{r}\alpha_{1,z} = \sin{}^{w}\psi_{r,x}\cos{}^{w}\theta_{r,z} \tag{6.64}$$

$$r_{32}\sin{}^{1}\beta_{2,x} + r_{33}\cos{}^{1}\beta_{2,x} = \cos{}^{w}\psi_{r,x} \tag{6.65}$$

where the r_{ij}, for $i = 1,2,3$ and $j = 1,2,3$, are the elements of the 3×3 rotation matrix $\mathbf{R}_c^{w}$.

Any three of the above equations can be used to compute the unknowns. For brevity only the results are presented here. First, the *tilt* angle of the camera is recovered from Equation 6.59 as follows:

$$\beta^{1}_{2,x} = \arctan\left(\frac{-r_{13}}{r_{12}}\right) \tag{6.66}$$

Second, the *pitch* angle is obtained by dividing Equation 6.62 by 6.65, the operation gives:

$$\psi^{w}_{r,x} = \arctan\left(\frac{-r_{22}\sin\beta^{1}_{2,x} - r_{23}\cos\beta^{1}_{2,x}}{r_{32}\sin\beta^{1}_{2,x} + r_{33}\cos\beta^{1}_{2,x}}\right) \tag{6.67}$$

Third, the *yaw* angle is computed in a similar fashion, dividing 6.58 by 6.57, then:

$$\theta^{w}_{r,z} = \arctan\left(\frac{-r_{11}\sin\alpha^{r}_{1,z} + \left(r_{12}\cos\beta^{1}_{2,x} - r_{13}\sin\beta^{1}_{2,x}\right)\cos\alpha^{r}_{1,z}}{r_{11}\cos\alpha^{r}_{1,z} - \left(r_{12}\cos\beta^{1}_{2,x} - r_{13}\sin\beta^{1}_{2,x}\right)\sin\alpha^{r}_{1,z}}\right) \tag{6.68}$$

where $\alpha^{r}_{1,z}$ is the *pan* angle of the camera, which is assumed known a priori as indicated in Section 6.4.2.

Care should be taken when interpreting the above equations, as each yields two possible solutions of opposite signs. To resolve the ambiguity, the sign of both arguments must be used to get the result in the correct quadrant.

Finally, once the *pitch*, $\psi_{r,x}^{w}$, and the *yaw*, $\theta_{r,z}^{w}$, angles are known, the position of vehicle with respect to the world coordinate frame, $\mathcal{F}_w$, can be directly computed using Equation 6.54.

A method for measuring the pose of the robot in a known environment has been presented in this section. The next task is to develop a pose tracking algorithm responsible for controlling the servo inputs.

6.6 Visual Servoing

In general, pose tracking of mobile robots that have a differential-drive configuration requires the manipulation of two inputs commands, the driving velocity, v_d, and steering velocity, ω_d, respectively. These two input commands should influence the three configuration variables (x, y, θ). Treating these three variables independently as in a typical control problem is nontrivial mainly because the vehicle base is underactuated [26].

The basic task that is considered is how to move the robot to a desired goal configuration, (x_d, θ_d), from a given initial pose $(x, y, \theta)^{\mathsf{T}}$ within the operational bound of the vision system. As formulated, the control objective makes no mention regarding the desired motion trajectory.

To achieve the navigational goal, a simple pose tracking controller is proposed. The controller uses a cubic spline to manipulate the desired steering velocity, ω_d, while the driving velocity, v_d, is externally specified. Both of these velocities are reference inputs to the low-level controller presented in Section 4.5.1. The polynomial uses the actual and desired configuration of the robot as boundary conditions. The control action is then derived from the curvature of the spline at its origin which is recomputed at each controller update.

6.6.1 Pose Tracking Controller

The pose tracking controller makes use of a special type of cubic curves called Cardinal splines [51]. Cardinal splines are easier to use than regular cubic splines because they do not require an explicit specification of the endpoint tangency conditions. Rather, the tangency at the endpoints is computed from two additional control points and a tension parameter. The mathematical derivation of the controller is shown below.

For a planar motion of the robot a path can be described by two parametric third-order polynomials, as follows:

$$\mathbf{S}(u) = \mathbf{A}u^3 + \mathbf{B}u^2 + \mathbf{C}u + \mathbf{D} \quad for \quad 0 \le u \le 1 \tag{6.69}$$

where

$$\mathbf{S}(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix}^T, \mathbf{A} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}^T, \mathbf{B} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}^T, \mathbf{C} = \begin{bmatrix} c_x \\ c_y \end{bmatrix}^T, \text{ and } \mathbf{D} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}^T.$$

A, **B**, **C**, and **D** are 1×2 row vectors of polynomial coefficients, and u is the curve parameter. Equation 6.69 may be represented in matrix equivalent form as follows:

$$\mathbf{S}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \end{bmatrix} \tag{6.70}$$

A solution for $\mathbf{S}(u)$ requires that eight constraints are specified. These can be obtained from the boundary conditions at the endpoints which are specified as follows:

$$\text{for } u=0: \quad \mathbf{S}(0)=\mathbf{p}_0,\ \mathbf{S}'(0)=\mathbf{T}_0 \quad \text{where} \quad \mathbf{T}_0=s(\mathbf{p}_1-\mathbf{p}_{-1})$$
$$\text{for } u=1: \quad \mathbf{S}(1)=\mathbf{p}_1,\ \mathbf{S}'(1)=\mathbf{T}_1 \quad \text{where} \quad \mathbf{T}_1=s(\mathbf{p}_2-\mathbf{p}_0)$$

where $s \geq 0$ is the tension parameter. The tension parameter is a characteristic of Cardinal splines. The coordinates $\mathbf{p}_{-1}$ and $\mathbf{p}_2$ are two additional control points introduced as a replacement for the tangency constraints $\mathbf{T}_0$ and $\mathbf{T}_1$ respectively. The spline curve does not pass through these points. Controlling the tangency at the endpoints is difficult as it affects not only the spline direction, but also the tangency rate in the vicinity of the endpoints.

The tangency of the cubic polynomials is obtained by taking the first derivative of Equation 6.70 with respect to its parameter u. This yields:

$$\mathbf{S}'(u)=\begin{bmatrix}3u^2 & 2u & u & 0\end{bmatrix}\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\\ \mathbf{C}\\ \mathbf{D}\end{bmatrix} \tag{6.71}$$

Solving Equations 6.70 and 6.71 simultaneously with the aforementioned constraints produces eight equalities that can be solved for the unknown coefficients as follows:

$$\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\\ \mathbf{C}\\ \mathbf{D}\end{bmatrix}=\begin{bmatrix}-s & 2-s & s-2 & s\\ 2s & s-3 & 3-2s & -s\\ -s & 0 & s & 0\\ 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{p}_{-1}\\ \mathbf{p}_0\\ \mathbf{p}_1\\ \mathbf{p}_2\end{bmatrix} \tag{6.72}$$

The matrix in the above equation is the Cardinal basis matrix. Substituting Equation 6.72 into Equation 6.70, gives:

$$\mathbf{S}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -a & 0 & a & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{-1} \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix} \tag{6.73}$$

In short notation Equation 6.73 is represented as follows:

$$\mathbf{S}(u) = \mathbf{U}\mathbf{M}_c\mathbf{G}_c \tag{6.74}$$

where

$$\mathbf{U} = \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}^{\mathsf{T}}, \quad \mathbf{M}_c = \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \text{ and } \quad \mathbf{G}_c = \begin{bmatrix} \mathbf{p}_{-1} \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}.$$

$\mathbf{M}_c$ is the Cardinal matrix and $\mathbf{G}_c$ the geometry vector. The particularity of Cardinal splines is that the basis of the cubic polynomial changes as its intrinsic tension parameter, s, is varied. Figure 6.7 depicts one such spline.

The figure shows the position of the control points $\mathbf{p}_{-1}, \mathbf{p}_0, \mathbf{p}_1$, and $\mathbf{p}_2$. The tangent at $\mathbf{p}_0$ is parallel to the vector $\mathbf{t}_1$. The tangent at $\mathbf{p}_1$ is parallel to the vector $\mathbf{t}_0$. The magnitude of the endpoint tangents, $\mathbf{T}_0$ and $\mathbf{T}_1$, is proportional to the length of these two vectors respectively. The spline endpoints $\mathbf{p}_0$ and $\mathbf{p}_1$ are chosen such that:

$$\mathbf{p}_0 = \begin{bmatrix} t^w_{r,x} \\ t^w_{r,y} \end{bmatrix} \tag{6.75}$$

which is the current position with respect to the world coordinate frame and:

$$\mathbf{p}_1 = \mathbf{p}_0 + \begin{bmatrix} e_x \\ e_y \end{bmatrix} \tag{6.76}$$

where e_x and e_y are the lateral and longitudinal error in position. In the current implementation of the controller $e_y = t^w_{r,y}$.

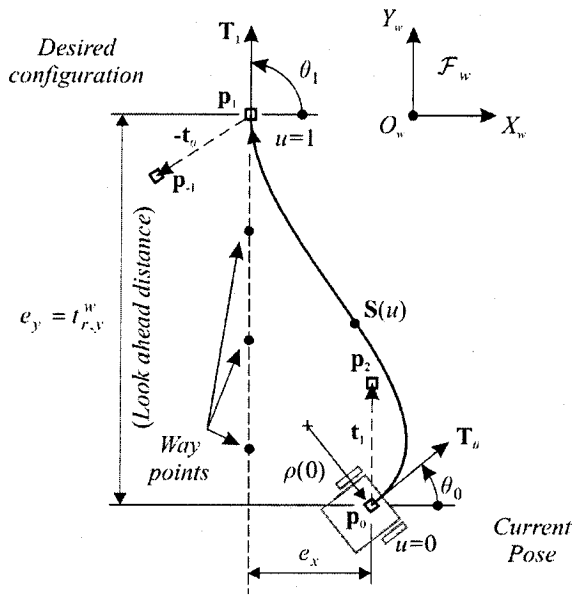


Figure 6.7: A Cardinal cubic spline. The curvature at the origin of the spline is used to control the steering velocity of the vehicle body.

The spline defined by Equation 6.70 has four degrees of freedom: the magnitude of $\|\mathbf{p}_1 - \mathbf{p}_{-1}\|$, the magnitude $\|\mathbf{p}_2 - \mathbf{p}_0\|$, the look ahead distance, $t^w_{r,y}$, and the tension parameter, s. The number of degrees of freedom reduces to two by further constraining the position of the control points $\mathbf{p}_{-1}$ and $\mathbf{p}_2$, such that:

$$\mathbf{p}_{-1} = \mathbf{p}_1 - \mathbf{t}_0 \tag{6.77}$$

$$\mathbf{p}_2 = \mathbf{p}_0 + \mathbf{t}_1 \tag{6.78}$$

where $\mathbf{t}_0$ and $\mathbf{t}_1$ are unit length vectors defined respectively by:

$$\mathbf{t}_0 = \frac{\cos\theta_0\mathbf{i} + \sin\theta_0\mathbf{j}}{\left\| \cos\theta_0\mathbf{i} + \sin\theta_0\mathbf{j} \right\|} \quad \text{and} \quad \mathbf{t}_1 = \frac{\cos\theta_1\mathbf{i} + \sin\theta_1\mathbf{j}}{\left\| \cos\theta_1\mathbf{i} + \sin\theta_1\mathbf{j} \right\|}.$$

It should be pointed out that the relations that exist between the tangency angles at the endpoints of the spline and the heading angle of the vehicle are respectively given by $\theta_0 = \pi + \theta_{r,z}^w\big|_{current}$ and $\theta_1 = \pi + \theta_{r,z}^w\big|_{desired}$.

The controller is left with only two degrees of freedom. The tension parameter, s, varies the tension of the spline while the look ahead distance, $t_{r,y}^w$, stretches the spline. They can be viewed as a controller tuning parameters analogous to proportional gains.

The spline curve itself is of little interest. Only the curvature at the origin of the spline, that is at $u = 0$, is required to derive the control action. The curvature at any position along the curve can be obtained from the following equation [52]:

$$\kappa(u) = \frac{\left\| \mathbf{S}'(u) \times \mathbf{S}''(u) \right\|}{\left\| \mathbf{S}'(u) \right\|^3} \tag{6.79}$$

where $\mathbf{S}(u) = x(u)\mathbf{i} + y(u)\mathbf{j}$ is a point along the curve. Performing the cross-product, an explicit expression for the curvature is obtained as follows:

$$\kappa(u) = \frac{x'(u)y''(u) - x''(u)y'(u)}{\left(x'(u)^2 + y'(u)^2\right)^{3/2}} \tag{6.80}$$

The radius of curvature of the spline at any point u on the curve is found as the reciprocal of the curvature as follows:

$$\rho(u) = \frac{1}{\kappa(u)} \quad \text{for} \quad \kappa(u) \neq 0 \tag{6.81}$$

Assuming the curvature at the origin is known, the steering velocity control law is computed as follows:

$$\omega_d = \kappa(0) v_d \tag{6.82}$$

where v_d and ω_d are the bounded driving and steering velocities corresponding to the nominal velocities v_d and ω_d respectively. The sign of the curvature, κ, obtained when evaluating Equation 6.80 at $u = 0$, indicates the steering direction. The desired velocities (v_d, ω_d) are the inputs of the low-level controller (refer to Figure 4.10).

A fast convergence to a desired configuration (x_d, θ_d) is possible by decreasing the value of the tension parameter, s, such that the curvature of the cubic spline at its origin is increased. However, the range for which the magnitude of the tension parameter yields stable control has not been investigated theoretically in this work.

Chapter 7

Experimental Results

"There are three principal means of acquiring knowledge... observation of nature, reflection, and experimentation. Observation collects facts; reflection combines them; experimentation verifies the result of that combination."

– Denis Diderot

Chapter 7

Experimental Results

7.1 Introduction

This chapter reports on the results of sets of experiments carried out on the vision-based navigational system. The navigational algorithm and methods presented in the previous chapters were tested using AUMER; a robotic vehicle implemented in the course of this research project.

First, Section 7.2 presents the methodology adopted to conduct the experiments. A complete description of the environment in which the navigation took place is included. The core of the chapter, Section 7.3, presents the results of two sets of experiments. Initially, the localization accuracy is assessed by comparing the vehicle pose obtained from the visual pose estimation algorithm and that derived from odometry with ground truth measurements. The results show how the localization accuracy can be improved by making advantage of redundancy of sensory signals and of sensory fusion. Then, a series of navigation experiments are performed to evaluate the performance of the pose tracking controller with respect to the stabilization of the vehicle heading orientation. Finally, Section 7.4 discusses the results with particular emphasis on issues related to image processing.

7.2 Methodology

The architecture of AUMER was presented in Chapter 3. The robot is equipped with two velocity feedback sensors, two optical encoders for odometry, and an onboard camera system for vision-based navigation. The image stream captured by the vision system is processed using a 2.08 GHz *AMD Athlon™ 2600XP+* processor that is also used for pose tracking and odometry data collection.

The camera is operated at a VGA resolution of 320×240 pixels^2, and the captured frames are transferred directly to the main processing board using 256 gray levels. The frame rate of the camera is controlled by the pose tracking controller. The current software implementation allows images to be captured at a maximum rate of two per second.

It is to be noted, that because the camera calibration was performed at a video resolution twice as large as the resolution at which the camera system is operated during the experiments, adjustment of the camera internal parameters was required. The effective pixel size being twice as large requires that each of the entries of the calibration matrix, $\mathbf{K}$, be divided by two to correct for the difference in video resolution.

7.2.1 Indoor Navigation

The corridor area where the navigation trials were conducted is 2 m wide and is assumed to extend indefinitely. The vehicle is constrained to move on a smooth level ground surface, free of any obstacle, and evenly distanced from the ceiling. As indicated earlier, ceiling features are used as natural landmarks to navigation and localization. The presence of a drop-in ceiling tile grid structure makes a good candidate for line feature extraction. More particularly, the grid structure is characterized by the two main runners

symmetrically distributed about the center of the corridor area. The runners are assumed parallel and are located 910 mm apart. This spacing makes the runners occupy an adequate proportion of the images captured. A larger spacing would make pose tracking difficult as line features would more easily move outside the field of view with only slight changes in heading orientation. On the other hand, a smaller spacing would reduce the accuracy associated with the vanishing point computation. Thus, the actual spacing offers a good compromise between both requirements.

The ceiling tile grid structure satisfies simultaneously the five conditions imposed by the geometric model presented in Section 6.3. The first condition requires that a minimum of two parallel straight lines are detectable by the camera. This condition is satisfied by the presence of two 12 mm wide main runners which are characterized by their low brightness. Because of this distinctive attribute, the line features can reliably be detected among other features of lesser interest. The second condition requires that the direction of the lines is known with respect to the robot environment. This constraint is fulfilled by the fact that the main runners are oriented parallel to the desired direction of motion. The third condition is satisfied by letting the camera point forward and tilting it upwards toward the ceiling such that the line features contained in the ceiling scenery are not parallel to the image plane. The tilt angle formed between the camera axis and the ground surface was adjusted to approximately 60°. The fourth condition relates to the orientation of the world coordinate frame. It is required for correctly identifying the rotation and translation components of the vehicle pose. The convention defined was strictly observed in the visual pose estimation algorithm. Finally, the distance separating the two main runners is known, 910 mm, thus satisfying the fifth condition. This distance is used as a metric for obtaining an estimate of the lateral position. The ceiling height could optionally be used. This distance was measured to 2 006 mm.

As an alternative to the ceiling scenery, the edges formed by the walls of the corridor with the floor or the ceiling could be considered equally good line feature candidates. The ceiling scenery, however, offers the advantage of easier localization of corridor centerline, and an uncluttered space which can remain unaltered for long time periods.

7.2.2 Ground Truth Measurements

To assess the performance of different pose estimation techniques on an operational robotic vehicle, a method of measuring the physical displacement of the robot had to be developed. One practical method is to trace the trajectory undergone by the robot. The method employed for this work involves a felt marker mounted on the robot and made to trace the robot trajectory on the ground. The marker was attached to the platform at the rear end at a distance of 0.28 m from the center of rotation and coincident with the platform axis of symmetry.

Following a navigation experiment, the marker trace left on the ground surface can be hand-measured at regular intervals and the collected measurements interpolated in software. However, because the marker tip is not collocated with the robot's center of rotation, (i.e., the midpoint of the wheelbase), a geometric transformation of the physical measurements is necessary.

Ground truth measurements are estimated to have an accuracy of ± 5 mm. The uncertainty associated with these measurements is mainly attributed to the thickness of the mark left on the ground surface, the deviation of the marker from the true axis of symmetry of the vehicle body, as well as the precision of the measuring device itself.

7.2.3 Performance Index

For the comparison of the localization accuracy of the different sensory fusion schemes with the actual trajectory followed by the robot, the following performance index is proposed:

$$J = \int_0^{\infty} |e_x| dy \tag{7.1}$$

where e_x is the error between the estimated and actual lateral position, x, and dy the estimated longitudinal displacement at each sampling intervals, T_s. The performance index, J, gives an indication of the localization accuracy by measuring the total absolute area between the actual and estimated trajectory providing a common basis for comparison.

7.3 Results

The results presented evaluate the localization accuracy and tracking performances of the pose tracking controller. The accuracy of localization is investigated by comparing the pose estimation obtained from different sensory fusion schemes. With regards to the pose tracking controller, the results show how the heading orientation can be stabilized using a simple controller structure that uses Cardinal cubic splines to derive the commanded steering velocity. This controller was described in Section 6.6.1.

7.3.1 Localization Accuracy

A first experiment was performed to evaluate the accuracy of localization using different sensors and their combination. For this experiment the pose tracking controller tension parameter, s, was set to 60, and the look ahead distance, $t^{w}_{r,y}$, was kept fixed at 1.2 m. To avoid saturation of the speed sensors during the vehicle motion, the vehicle driving velocity, v, was set to

0.23 m/s, which corresponds to approximately 90 % of the robot maximal driving speed. The selection of the tension parameter is such that appreciable deviation of the heading orientation occurs while performing the experiment so as to amplify the effect on the different methods of sensory data collection.

During the experiment three sets of measurement were collected. The heading angle, $\theta^{w}_{r,z}$, and the lateral position, $t^{w}_{r,x}$, of the vehicle body were obtained with the visual pose estimation algorithm. The angular velocity, $\dot{\varphi}_{r,l}$, of the left and right-hand wheels were recorded from the velocity feedback sensors. Finally, the pulse counts from the optical encoders, mounted on each of the two wheel drive actuators were used to determine the angular position of the wheels, $\varphi_{r,l}$, as a function of time.

The odometry data sets recorded were preprocessed by removing outliers and filtering noise using a first-order low-pass filter. Filter constants ranging from 0.6 to 0.7 showed good noise attenuation without introducing significant phase shifts in the measurements. Because not all measurements were recorded at the same frequency, resampling of the data sets was required. This is a direct consequence of the different sampling rate associated with the high and low-level of control.

The pose estimates presented here are grouped into two categories, depending on whether it is derived from the velocity feedback sensors or directly from the optical encoders. For each of the two categories, three sensory fusion schemes are evaluated. First, the pose estimates are obtained from the integration of incremental motion (i.e., pure odometry). Second, the pose estimates are obtained by updating at regular intervals the odometry measurements with an absolute measure of the heading angle as measured with the visual pose estimation algorithm. Third, the pose estimates are obtained by updating the odometry simultaneously with an absolute

measurement of the heading angle and of the lateral position as obtained from the visual pose estimation algorithm. The results are presented in Figures 7.1 and 7.2. Tables 7.1 and 7.2 list the performance indices obtained for the various sensory fusion schemes.

Figure 7.1 shows the experimental results for the case of using the odometry data collected with the optical encoders to estimate the localization of the robot. For each of the three sensory fusion schemes presented above, the localization variables are calculated using Equations 4.87 to 4.89. The estimated trajectories are then compared to the actual course taken by the robot during the navigation trial. It is to be noted that the graph abscissa and ordinate axes differ in scale to emphasize the discrepancies of the different sensory fusion schemes.

Figure 7.1 and Table 7.1 show that when using pure odometry, the error on position increases with traveled distance and rapidly becomes too high to offer a reliable estimation of the vehicle pose. Because, the pose estimates are derived from a discrete integration process, any errors in incremental motion estimates accumulate over time. A deterioration of the localization accuracy quickly results. These errors usually originate from wheel slippage, inaccuracy of the wheelbase dimension, and varying wheel diameter due the vehicle body weight. These types of errors are common cause of odometry drift and have been widely acknowledged in the literature [4], [53].

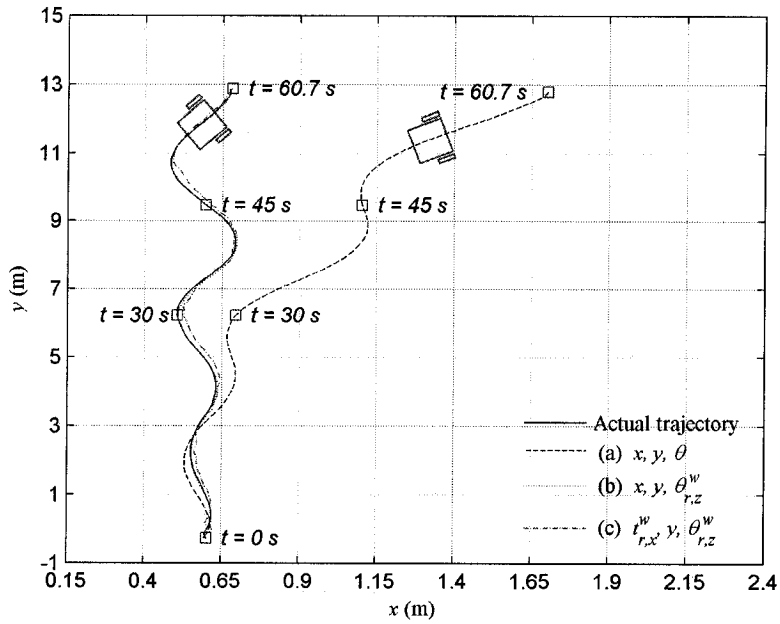


Figure 7.1: Vehicle pose estimated from the integration of the left and right-hand wheel angular positions. (a) pure odometry, (b) pose estimates obtained in (a) are updated at regular intervals with an absolute measurement of the heading angle, and (c) pose estimates obtained in (b) are updated with an absolute measurement of the lateral position. Update time interval: 0.065 s.

Table 7.1: Performance indices for the pose estimates obtained from the integration of the wheel angular positions.

Odometry estimates	Visual pose estimates	Performance Index, J	Distance Traveled[a] (m)
x, y, θ	—	4.02	13.19
x, y	$\theta^{w}_{r,z}$	0.07	13.19
y	$t^{w}_{r,x}, \theta^{w}_{r,z}$	0.22	13.19

[a] Actual distance traveled: 13.20 m.

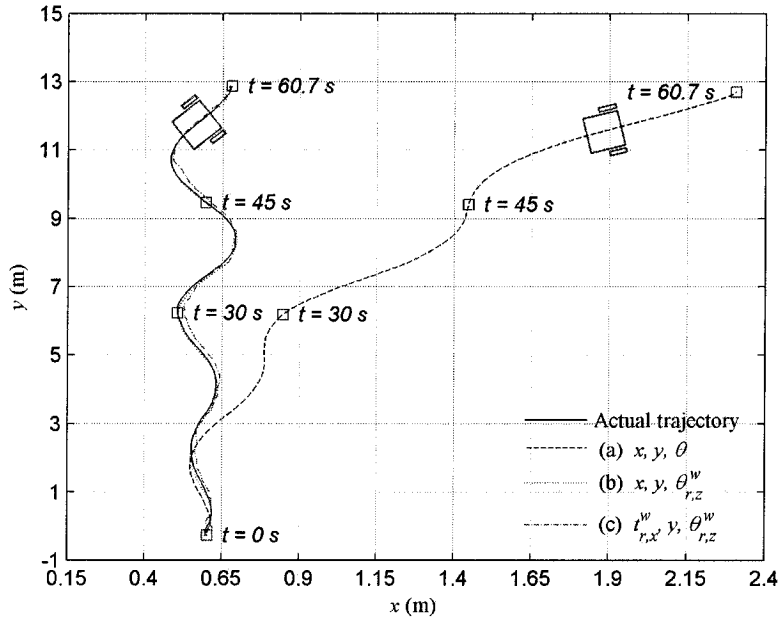


Figure 7.2: Vehicle pose estimated from the integration of the left and right-hand wheel angular velocities. (a) pure odometry, (b) pose estimates obtained in (a) are updated at regular intervals with an absolute measurement of the heading angle, and (c) pose estimates obtained in (b) are updated with an absolute measurement of the lateral position. Update time interval: 0.065 s.

Table 7.2: Performance indices for the pose estimates obtained from integration of the wheel angular velocities.

Odometry estimates	**Visual pose estimates**	**Performance Index, J**	**Distance Traveled**[a] (m)
x, y, θ	—	6.67	13.20
x, y	$\theta^{w}_{r,z}$	0.09	13.20
y	$t^{w}_{r,x}, \theta^{w}_{r,z}$	0.23	13.20

[a] Actual distance traveled: 13.20 m.

As the performance index results listed in Table 7.1 demonstrate, the pose estimation can significantly[1] be improved by updating at regular intervals the odometry estimates with a measurement of the heading angle such as that computed with the visual pose estimation algorithm. Contrary to pure odometry integration, this measure is absolute, thus not sensitive to drift.

Comparable accuracies are obtained when updating the odometry estimates from the velocity feedback sensors simultaneously with an absolute measure of the heading angle and of the lateral position. However, noticeable noise in the pose estimates is present from this sensor combination scheme. This is because the lateral position is obtained from a plane-to-plane homography between the camera plane and the ceiling features, thus is more sensitive to the image spatial resolution.

The second figure, Figure 7.2, shows the pose estimates obtained when integrating the wheel angular velocities as an alternative to the wheel angular positions. For that particular case, Equations 4.81 to 4.86 are used for estimating the localization variables. The trajectory estimated from the integration of the wheel angular velocities shows the same trend as that obtained from the integration of the angular position. However, the estimates obtained from the velocity feedback sensors seemed to drift faster than those derived from the optical encoders. A possible explanation for that is attributed to the difference in the physical characteristics of the wheel drive actuators which was not noticed when calibrating the velocity feedback sensors a first time using a stroboscope.

---

[1] Provided that ground contact of the wheels is maintained at all time.

This was confirmed by recalibrating the velocity sensors using the signal read from the optical encoder as an absolute reference. As this observation shows, the small discrepancies that subsisted between the sensed rotational speed of the wheel and the actual one, rapidly caused the odometry estimates obtained from the velocity feedback sensors to deteriorate. The error introduced systematically affects the computation of the actual steering velocity. For the subsequent experiments, the physical difference of both drive actuators was corrected by adjusting the modulation frequency of the wheel drive actuator drivers such that the odometry estimate obtained from the velocity feedback sensors correlate to that derived from the optical encoders.

A deeper insight into the possible cause of odometry drift, led to examining the effects of poorly estimated heading angles. With reference to Equation 4.87, it can be seen that the heading angle is obtained from a differentiation process. Differentiation of a sensed signal amplifies noise which when integrated over time may become an additional source of drift in the estimation of the heading angle. Being dependent on the heading angle, a fast degradation in the estimation of the position variables, x and y, consequently follow. A comparison of the estimated heading angles as a function of longitudinal displacement, y, is shown in Figure 7.3.

The graph shows a fast degradation in the estimate of the heading angle obtained from odometry integration when compared to the heading angle obtained from the visual pose estimation algorithm. The figure shows how the effects of wheel angular velocity calibration errors lead to a drift in the estimation of the heading angle that is almost twice as large as the drift experience when using the wheel angular positions.

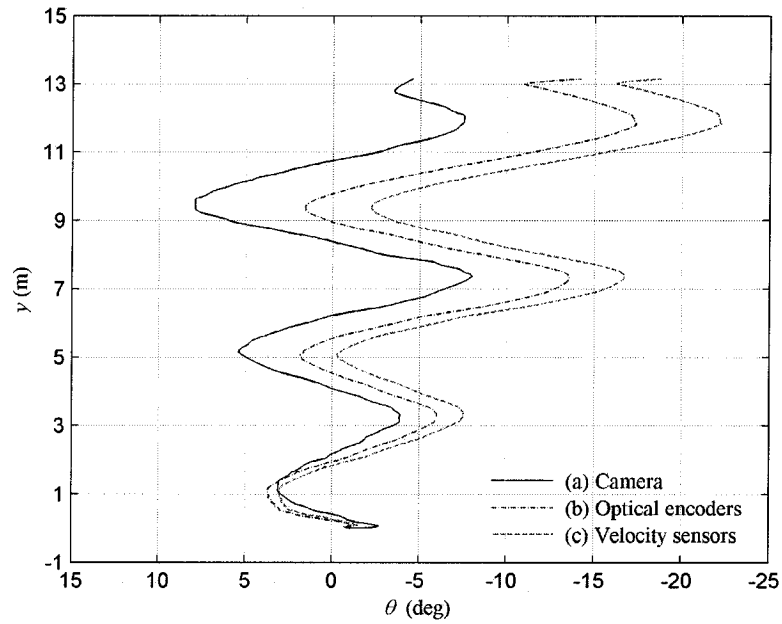


Figure 7.3: Heading angle estimates. (a) actual heading angle as measured by the visual pose estimation algorithm, (b) estimated heading angle obtained from the integration of the left and right-hand wheel angular position using Equation 4.87, and (c) estimated heading angle obtained from integration of the left and right-hand wheel angular velocities using Equation 4.81.

In summary, six sensory fusion schemes have been compared: three estimating the pose by integrating the wheel angular velocities, and three others by integrating their angular positions over time. The results presented show that the reliability of the pose estimation can be improved by updating the odometry estimates at regular intervals with absolute measurements of the pose variables.

Estimating the pose using the optical encoder readings seemed to be the more accurate of the six sensory fusion schemes presented as indicated by the performance index. However, all schemes that utilized an absolute

measurement, the heading angle or the lateral position, yielded comparable localization accuracy.

Estimating the velocity feedback sensors has the potential to be as accurate as that derived from the optical encoders assuming the heading angle can be updated at regular intervals from an external source. However, as was noted, it is prone to calibration errors with respect to the wheel angular velocities.

7.3.2 Pose Tracking Controller Performance

This section presents three navigation experiments conducted to evaluate the performance of the pose tracking controller with regard to the stabilization of the heading orientation of the vehicle. For that purpose, the pose tracking controller proposed in Section 6.6.1 was implemented. The controller uses Cardinal splines to derive the desired steering velocity. This is achieved by recursively computing a spline between the actual pose of the robot and a fictitious dynamic way point located at a fixed look ahead distance, along the direction of motion. The curvature of the spline at its origin is used in the computation of the desired steering velocity of the vehicle body.

For each of the three experiments, the tension parameter of the spline, s, was varied. Varying the tension parameter, s, increases or decreases globally the curvature of the spline. The effects of the tension parameter on the steering velocity as well as the robot trajectory were experimentally observed while keeping constant the look ahead distance, $t_{r,x}^{w}$.

The results of the experiments are presented in Figure 7.4 to 7.6. The figures show the actual trajectory of the vehicle body when commanded to follow a rectilinear trajectory, at a constant driving, $v = 0.23$ m/s, tracking ceiling line features. Figure 7.4 shows the trajectory obtained for the case of

a tension parameter, s, set to 60. Oscillation in the trajectory is observed to lead to an unstable behavior of the vehicle body.

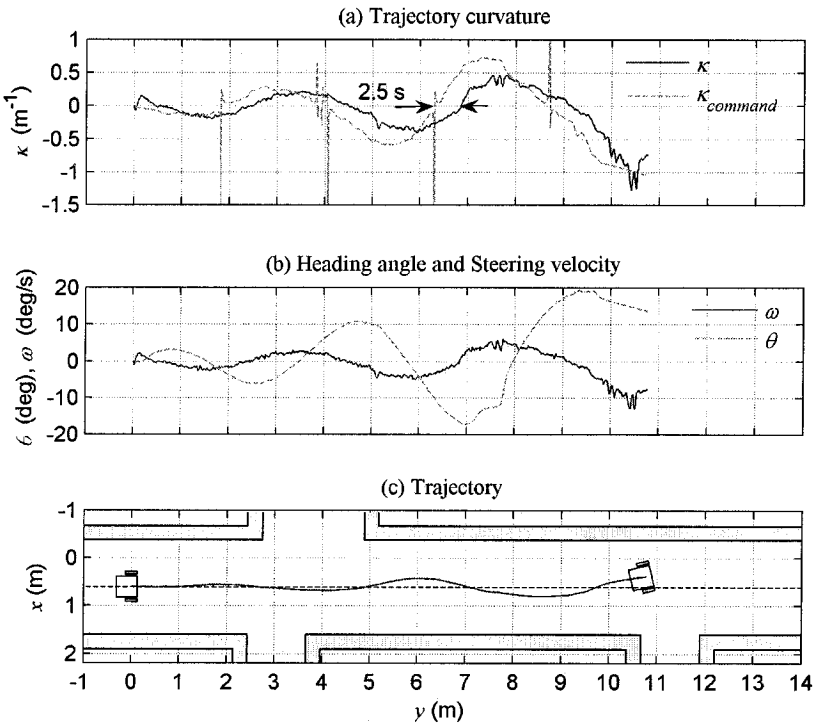


Figure 7.4: Navigation experiment no. 1: s = 60. (a) trajectory curvature, (b) heading angle and steering velocity, and (c) actual in-plane trajectory.

The test failed after having traveled 10.88 m due to large fluctuations in the heading angle, $\theta = \pm 20°$, because the line feature used to infer the pose of the vehicle moved out of the field of view. This is a direct consequence of having the camera rigidly attached to the vehicle body.

The tracking of ceiling line features can be improved by increasing the tension parameter to $s = 75$ such that the tension of the spline is decreased. Because, increasing the tension parameter is equivalent to decreasing the

desired steering velocity of the vehicle body, ω_d, the magnitude of the oscillations is diminished. The results obtained are presented in Figure 7.5.

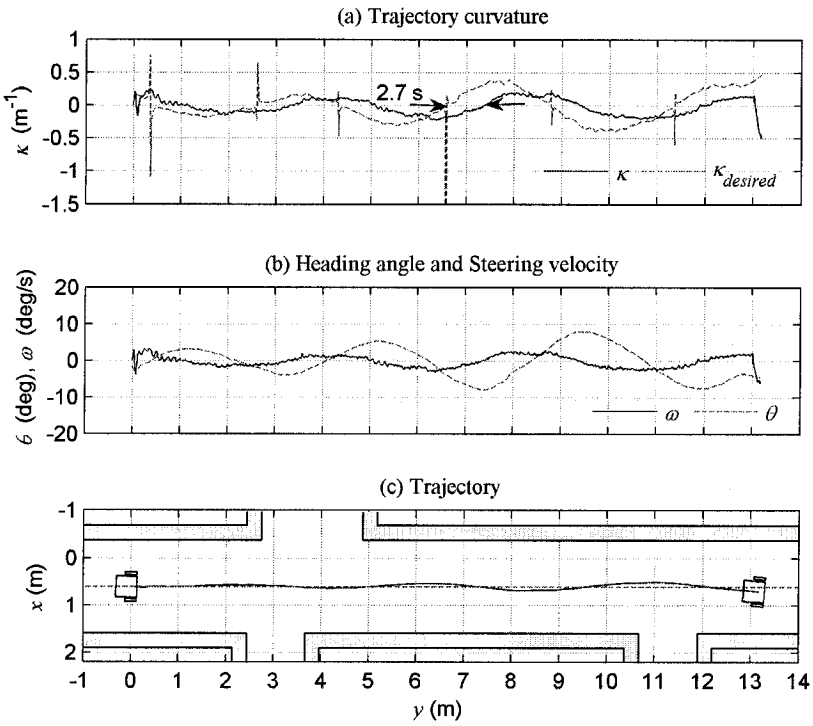


Figure 7.5: Navigation experiment no. 2: s = 75. (a) trajectory curvature, (b) heading angle and steering velocity, and (c) actual in-plane trajectory.

Increasing the tension parameter from 60 to 75 reduced the magnitude of trajectory oscillations with a heading angle varying between $\theta = \pm 10°$. Yet, signs of divergence are perceivable, particularly at the end of the trajectory.

For the last experiment the tension parameter was set to $s = 90$. The results presented in Figure 7.6 show that increasing the tension parameter makes the trajectory oscillation vanish for the entire traveled distance.

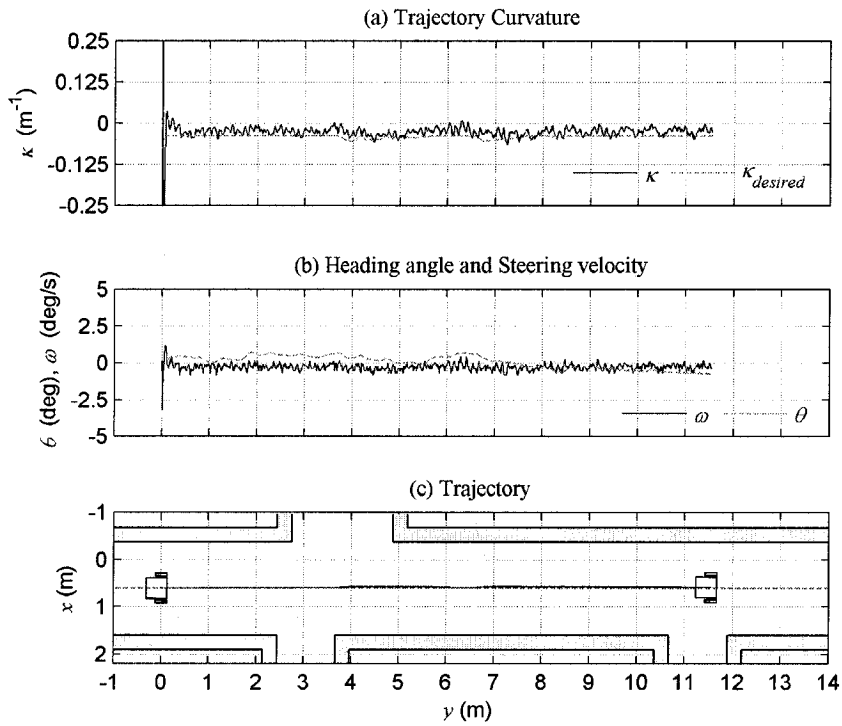


Figure 7.6: Navigation experiment no. 3: s = 90. (a) trajectory curvature, (b) heading angle and steering velocity, and (c) actual in-plane trajectory.

However, special attention is due to the desired curvature set point shown in Figure 7.6(a). For the duration of the navigation trial, the desired curvature input remained almost constant, fluctuating around -0.03 m^{-1} while the heading angle has fluctuated between $\pm 1°$. This suggests that the heading angle error has a weak effect on the current controller structure, especially, for high tension parameter values.

It should be noted that for all three experiments, the actual trajectory curvature shows a significant phase shift compared to the desired curvature input. Part of this phase shift is attributed the dynamic behavior of the vehicle body. However, because the wheel actuators are capable of delivering high-torque due to their low gearbox transmission ratio, the combined effects of mass inertia and wheel actuator inductances are greatly attenuated. The unusual magnitude of the lag suggests that the phase shift is introduced by some other sources. A highly probable cause is the latency introduced by the image processing algorithm. The consequence of this latency is that the communication between the high and low control levels is delayed. For the set of experiment presented here this delay is estimated to be 0.5 s.

For comparing the performance of the controller with respect to the spline tension parameter, Figure 7.7 graphs the performance index for the first 51 s of navigation. The index was defined in Equation 7.1.

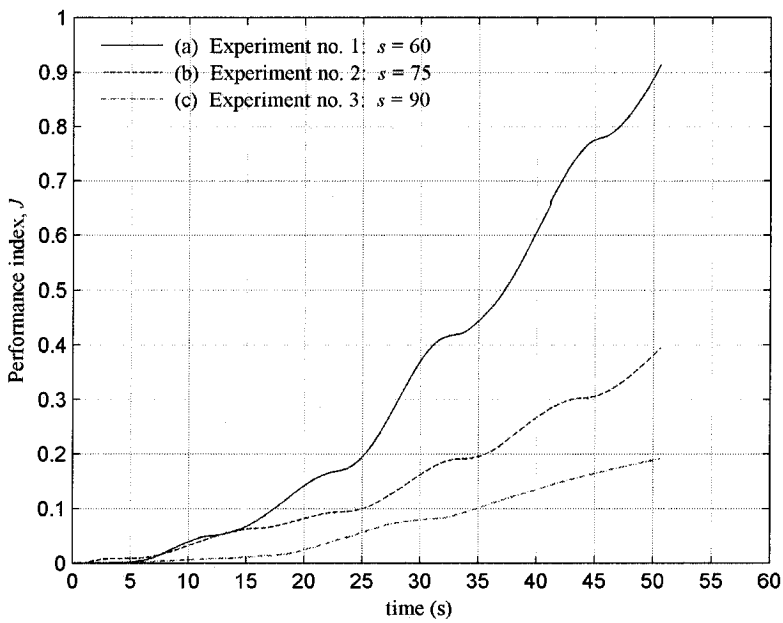


Figure 7.7: Performance index given as a function of time. (a) Experiment no. 1, (b) Experiment no. 2, and (c) Experiment no. 3.

None of the tests conducted clearly show evidence of a perfect tracking of the desired lateral position. This is particularly true for the case of Experiments no. 1 and 2 whose tension parameters were set to $s = 60$ and $s = 75$ respectively. This is due to the incapacity of the pose tracking controller to asymptotically make the system reach the desired heading angle. However, increasing the spline tension parameter to $s = 90$, as in Experiment no. 3, caused the system to better track the heading angle. As a result, the magnitude of the integral of the absolute error at the end of the course is about five times less than that obtained in Experiment no. 1, and two times less than that obtained in Experiment no. 2. Furthermore, because the slope of the curve obtained for Experiment no. 3 is decreasing, it suggests that perfect tracking of the lateral position was eminent at the time the navigation trial was stopped.

In summary, this section has presented a set of experiments that demonstrate how Cardinal splines can be utilized to command the steering velocity of a differentially-driven robotic vehicle. The results show that the magnitude of oscillation of the vehicle body can be reduced by decreasing the curvature of the spline utilized to derived the control law via its tension parameter, s, a characteristic of Cardinal splines. However, decreasing the curvature led to a system with a slow responsiveness in reaching a desired pose. Abnormal phase shift present between the command input and the response of the system has been identified. This is believed to have a direct effect on the stabilization of the heading orientation of the vehicle.

7.4 Discussion

The experiment results presented here are those of successful navigation trials, however, not all navigation trials have been successful. A number of failure causes have been identified. These failures are mostly attributed to image processing and the problem associated with line feature extraction.

Image Artifacts — Artifacts in the image were the most notorious difficulties associated with the image processing algorithm. Artifacts are considered here as any dark region (i.e., low brightness) in the captured image other than line features contributing to the dominant vanishing point. The artifacts primarily originate from binarization; a process that converts an image from gray levels to a bilevel, (i.e., black and white). The binarization process immediately follows the image capture, and is used to isolate the two main runners from the ceiling scenery based on a fixed threshold value determined by their intensity range. However, because the vehicle is constantly moving, other scene objects having the same intensity range may enter and leave the field of view as the vehicle maneuvers. Door frames and shadows are two common examples. Their presence in the image hinders the detection of the good line features. Increasing the brightness of the captured image or the content of white light, and adjusting the camera gain prior to binarization partially remedied this problem.

Scene Illumination — The presence of bright sources of light in the image scene also caused a deterioration of the images but not as much as was anticipated. In fact, proper lighting was beneficial for contrasting lines from other scene features. Nevertheless, localized sensor saturation, blooming effect, has caused the partial disappearance of good line features in the close vicinity of the sources. To some extent, this problem was corrected by adjusting the exposure time before each navigation trials.

Feature Tracking — Losing track of the line features used for estimating the pose of the vehicle as it maneuvered has also been identified as the most notorious cause of failure. This is particularly true since the camera is rigidly fixed to the robot chassis. This problem can be avoided by stabilizing the heading orientation of the vehicle as it maneuvers via the proper adjustment of the pose tracking controller, namely the spline tension parameter. Another approach would be to increase the field of view of the camera by replacing the actual lens with a lens of a shorter focal length.

Vanishing Point — With regards to the vanishing point analysis and camera orientation, fluctuation in the measurement of the camera tilt angle has been noticed. In the ideal case, because the camera tilt angle was kept fixed during the experiments the vanishing point should be found at a constant v-coordinate on the image plane usually referred to the horizon line. However, as Figure 7.8 shows its position fluctuates as the vehicle maneuvers. Since the computation of the camera heading angle and lateral position are functions of the camera tilt angle, the uncertainty which results from this measurement propagates through their computation as well.

These fluctuations are introduced as a result of the course discretization of the Hough space. A finer discretization, would certainly decrease the uncertainty associated with this measurement and reduce noise-like fluctuation in the estimation of the vehicle pose. However, it would lead to an increase in image processing time.

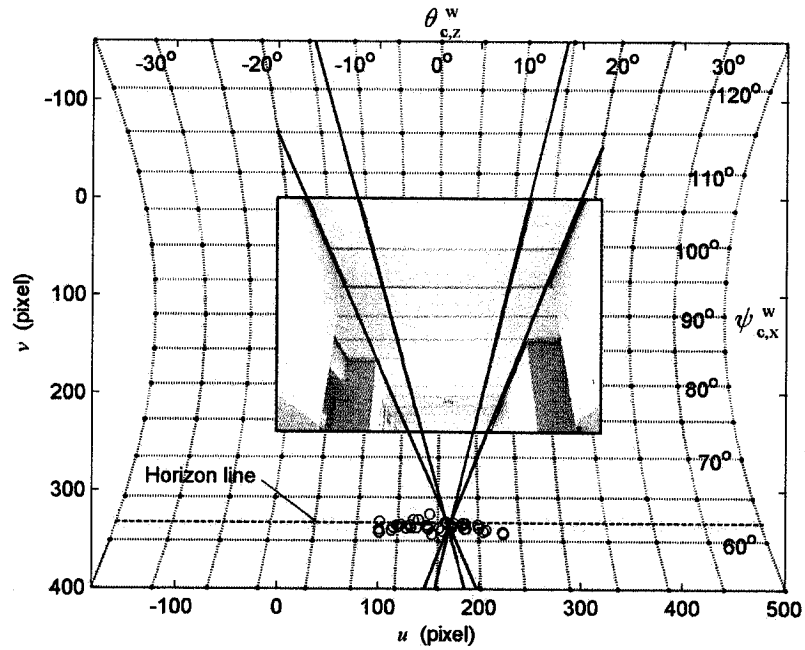


Figure 7.8: Typical locations of the vanishing point as sampled during a navigation experiment. For this particular case, the camera tilt, $\beta^{1}_{2,x}$, is found to be $62.1° \pm 0.88°$. Only a subset of the samples is shown for clarity.

Planar Homography — Uncertainty in the measurement of the lateral position is also introduced by the fact that only two correspondence pairs are used to compute the plane-to-plane homography. In practice, planar homographies are computed from more correspondences than are actually required such that the unknowns are estimated from a least-square minimization. In the current context, this approach has been avoided such that the visual pose estimation algorithm stays generic and weakly coupled to the physical environment. Still, the proposed algorithm requires that a metric about the environment is known a priori, should that be the height of the ceiling or the length of a ceiling tile. As an alternative to computing a planar homography, the lateral position could be obtained from the integration of the left and right-hand wheel angular positions, and that

without loss of accuracy as the experimental results clearly demonstrated (refer to Table 7.1 and 7.2).

Controller Structure — Further investigation of the proposed controller structure is required. For instance, the effects of the look ahead distance, are not fully understood at this stage. The current controller possesses a discontinuity in its outputs. The discontinuity is introduced in the computation of the cardinal spline radius of curvature used to derive the desired steering velocity. Figure 7.9 shows a plot of the radius of curvature at the origin of the spline ($u = 0$) as a function of the tension parameter and of the look ahead distance.

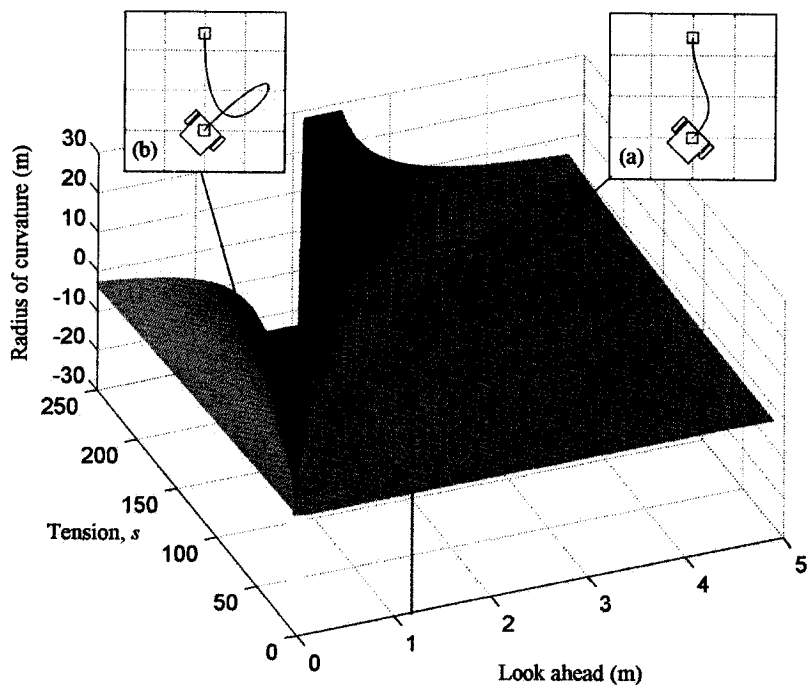


Figure 7.9: Radius of curvature at the origin of the spline ($u = 0$) as a function of the tension parameter, s, and of the look ahead distance. (a) spline of proper curvature sign at its origin, and (b) spline loops back on itself which causes curvature sign inversion.

For the simulation, the lateral position, and the heading angle have been kept constant to 0 m and 45° respectively. A similar discontinuity, but at different locations, can be observed when these parameters are varied. The Figures 7.9(a) and 7.9(b) illustrate the behavior of the spline in the two distinct regions delimited by the discontinuity.

The region indicated by Figure 7.9(a) is the intended region of operation while the region identified by Figure 7.9(b) illustrates an undesirable behavior which leads to a sign inversion of the radius of curvature and spline looping. As it appears, the tension parameter and the look ahead distance relate to each other. Selecting both independently, as was the case in the experiments presented, can potentially add an additional source of instability during online navigation as the actual heading angle and lateral position affect the end conditions at the origin of the spline. An improper selection of the look ahead distance and of the tension parameter may cause the controller to operate in both regions.

Chapter 8

Conclusion

"A conclusion is the place where you got tired of thinking."

– Arthur Bloch

Chapter 8

Conclusions

8.1 Concluding Remarks

The research presented in this thesis has addressed the problem of localization and motion control of an autonomous mobile robotic vehicle. The methods and techniques developed in this work have led to several conclusions as outlined below.

8.1.1 Visual Pose Estimation Algorithm

In this thesis a method for estimating the position and orientation of a mobile robot has been developed. The method uses visual clues from a robot's environment and projective geometry techniques to extracts its orientation and lateral position in a two-dimensional space.

The orientation is recovered using a method developed by Schuster [21] and heavily relies on the use of vanishing points. The method requires to track the location of the dominant vanishing point from a stream of images. Because a vanishing point is independent of translation, finding its location onto the image plane allows to recover the orientation of the vehicle independently from its position.

The lateral position is recovered with an original technique which uses hypothetical homography correspondences. The method is distinguished from other techniques in that it does not rely on marker detection techniques nor does it require visual modifications to the navigation environment. Rather, it uses line features – passing through a vanishing point – to identify the two hypothetical correspondences required for solving an homography problem. Minimally, only two line features need be identified.

The proposed approach has some limitations. The current implementation of the visual pose estimation algorithm only discerns two components of the pose vector: the orientation, θ, and the lateral position, x. The longitudinal position, y, must be estimated using odometry integration or other means. In addition, the technique requires an accurate model of the camera sensor and an internal representation of the robot's environment.

The visual pose estimation algorithm has been fully implemented on **AUMER**. For the purpose of comparison and because of the redundancy of the sensory inputs, pose estimation has been evaluated for six sensory fusion schemes. Three of the schemes evaluated were based on the integration of the wheel velocities, and three others based on the integration of the wheel angular positions. Although the idea of fusing sensory information to augment the reliability of the localization variables is not new, directly fusing the odometry data with visual pose estimates seems to be original.

The result presented clearly demonstrates the potential of the approach. Deriving the pose estimates from the integration of the wheel angular positions with regular updates of the heading orientation has proven to be the more accurate of the six sensory combination schemes presented as per the performance index, J. Other sensory fusion schemes that utilized a visual

pose estimates, should that be the heading angle, or the lateral position, yielded comparable localization accuracy.

8.1.2 Motion Control

This research work has presented in Chapter 7 an original pose tracking method. The originality of the proposed controller structure resides in the fact that it utilizes the geometrical property of Cardinal splines to generate the steering velocity and control a robot's motion.

The controller was implemented and tested in a real indoor environment by specifying rectilinear paths though the specification of successive dynamic waypoints. The experimental evaluation has focused on the effect of the spline tension parameter, s, on the stabilization of the heading orientation. Three experiments have been conducted which permit to conclude that decreasing the curvature of the spline by adjusting its tension parameter stabilizes the heading orientation but also leads to slow responsiveness. Conversely, increasing the curvature leads to a fast response, which is highly desirable, but also leads to prohibitive oscillation of the heading orientation. Heading angles outside the range $\theta = \pm 20^{\circ}$ with respect to the desired rectilinear path led to navigation failure.

Further investigation of the proposed controller structure is required. For instance, the effects of the look ahead distance on the controller output is not fully understood at this stage. Quantifying the distinctive contribution of the lateral position and heading angle error as well as their effect on the steering velocity is also an area that commands an inquiry.

8.2 Summary of Contributions

This research contributes to the general body of knowledge in the areas of mobile robotics and image analysis. The contributions of this thesis are outlined below in the order of scientific impact.

An Accurate Visual Pose Estimation Algorithm — Wheel encoder odometry methods are poor proxies of the heading orientation and result in a diverging error in localization. A major goal of this work has been to develop a vision-based localization technique to overcome this limitation. A standard technique for recovering a camera orientation using scene vanishing points is reformulated. This work extended the technique one step further to include the recovery of the lateral position. A closed-form expression for the lateral position was obtained by setting up a hypothetical homography problem.

The main advantage of the method lies in the fact that it only requires three pieces of information – two parallel lines and a metric – to correctly identify two of the localization variables. Implementation of the visual pose estimation algorithm has proven its superiority with regards to localization accuracy when compared with pure odometry. Furthermore, the method is generic and can be applied to a wide range of indoor environments.

An Automated Strategy for Tracking Vanishing Points — A key element of this thesis has been image analysis and line feature extraction. In Chapter 6 an image analysis algorithm for finding the image coordinates of a vanishing point using the Hough transformation was presented. The analysis of the Hough space reveals that line features directly contributing to a vanishing point form a special pattern of peaks in that space. This allows the extraction of only a specific class of line features, rather than have the

extraction algorithm blindly search for the relevant ones. This approach is believed to be a promising avenue for future research in the field of image analysis and vanishing point extraction.

A Complete Mathematical Model of the Vehicle — Chapter 4 has presented a dynamic model of the robotic vehicle. This model has been used to synthesize the velocity feedback control law of the wheel-drive actuators. Such a model can be used as a launching pad toward the development of a complete navigation simulator which would allow to experiment control strategies and navigation algorithms offline.

Goal-Directed Navigation and Motion Control — This thesis also contributes to the work on motion control of nonholonomic robots by demonstrating an original pose tracking method. The method uses the geometrical properties of Cardinal splines to control a robot's motion – an approach that has not been developed before. The controller was implemented and tested in a real indoor environment. The necessary points for defining the Cardinal spline are generated dynamically through waypoints gleaned from the image analysis data. The current pose controller evaluation is a prelude to an in-depth theoretical analysis of the controller efficacy.

A Full Implementation of a Mobile Robotic Vehicle — AUMER is a fully functional mobile robotic platform implemented for testing the navigational algorithms presented in Chapter 6. The hardware details provided in Chapter 3 and source code found in the appendices will enable researchers to build upon the current implementation.

8.3 Future Work

The main focus of this work has been the development of a fully operational mobile robot architecture using artificial vision. Although the environment in which the tests were performed and the tasks executed were both limited in scope, successful vision-based navigation has been demonstrated. Above all, these navigation trials permitted the identification of areas where improvement is required before useful autonomous navigation in larger-scale environment can be considered. The present section proposes three main areas where future work should be directed.

8.3.1 Vehicle Architecture

Losing track of the line features used for estimating the pose of the vehicle as it maneuvers has been identified as an important source of failure. This is particularly true since the camera is rigidly fixed to the robot chassis. This problem can be avoided by stabilizing the heading orientation of the vehicle as it maneuvers via the proper adjustment of the pose tracking controller, namely, the tension parameter. However, this restricts the orientation of the vehicle body to stay within $\pm 20^{\circ}$ from the desired heading.

As a complement to the current design, a pan-tilt head mechanism is certainly ranked one of the highest priority. Such a mechanism would not only more easily keep track of visual features used to infer localization but also physically decouple the rotational movement of the camera from the rotation of the robotic vehicle body. The apparent dynamic behavior of the object contained in the scene would significantly be reduced, therefore minimizing motion blurs.

In terms of control, vanishing point and a pan-tilt head seems to be an ideal combination. As was shown in this work, the image coordinates of the

vanishing point directly relate to the orientation of the camera, and are independent of translation. The two axes of the head mechanism, the pan and tilt, could therefore be controlled using two position servos whose input signal would be derived from the error between the actual and desired vanishing point location in the image plane.

With regard to localization, the result showed that the accuracy can significantly be improved by updating the odometry at regular intervals with an absolute measure of the heading orientation. However, this was achieved under the assumption that wheel slippage does not occur. In practice, this is seldom the case, and an additional way of sensing the localization variables, one not affected by wheel slippage, might be required.

To that end, the possibility of using dedicated optical flow motion sensors, as an additional mean for sensing the localization variables is worth further investigation. Optical flow sensing is a mature technology which is utilized in optical mice. In-plane motion is detected by comparing sequential images captured by a low-resolution camera, 18×18 pixels^2, using a digital signal processor. However, typical optical mice have very short focal length lenses which require their close proximity to the viewing target plane. This limitation could easily be overcome by replacing the lens with a longer focal length. One advantage of this technique, is that the x and y components of motion could directly be polled from the Main Processing Board without further need for a separate hardware interface, as this capability is already built-in.

Obstacle detection and avoidance is a research problem yet to be addressed. The current implementation does not provide a mean to that effect. Although vision could be used, sonar sensors have proven their efficiency for this particular task and are readily available commercially.

Augmenting the sensing capability of the current platform for obstacle detection would only require minimal development overhead, while preserving the limited computational resources to higher-level vision-based cognitive tasks such as places recognition and topological mapping.

8.3.2 Line Feature Extraction

The reliability of the image processing is a requirement that cannot be understated. As it has been noted, navigation depends on the successful extraction of line features. In the current implementation of the navigational algorithm, the Hough transform is a key method for the detection of line features contained in the image. This is achieved by transforming the image into a secondary domain, the Hough space, wherein the local maxima's coordinates indicate the parameters of the line feature identified. Searching for these maxima is currently performed by systematically scanning the entire domain, leading to the extraction of line features of lesser interest as well as adding to the computational load.

However, the pattern formed by the local maxima corresponding to line segments that contribute to a vanishing point suggests that a more efficient method could be utilized. Figure 8.1 illustrates this claim.

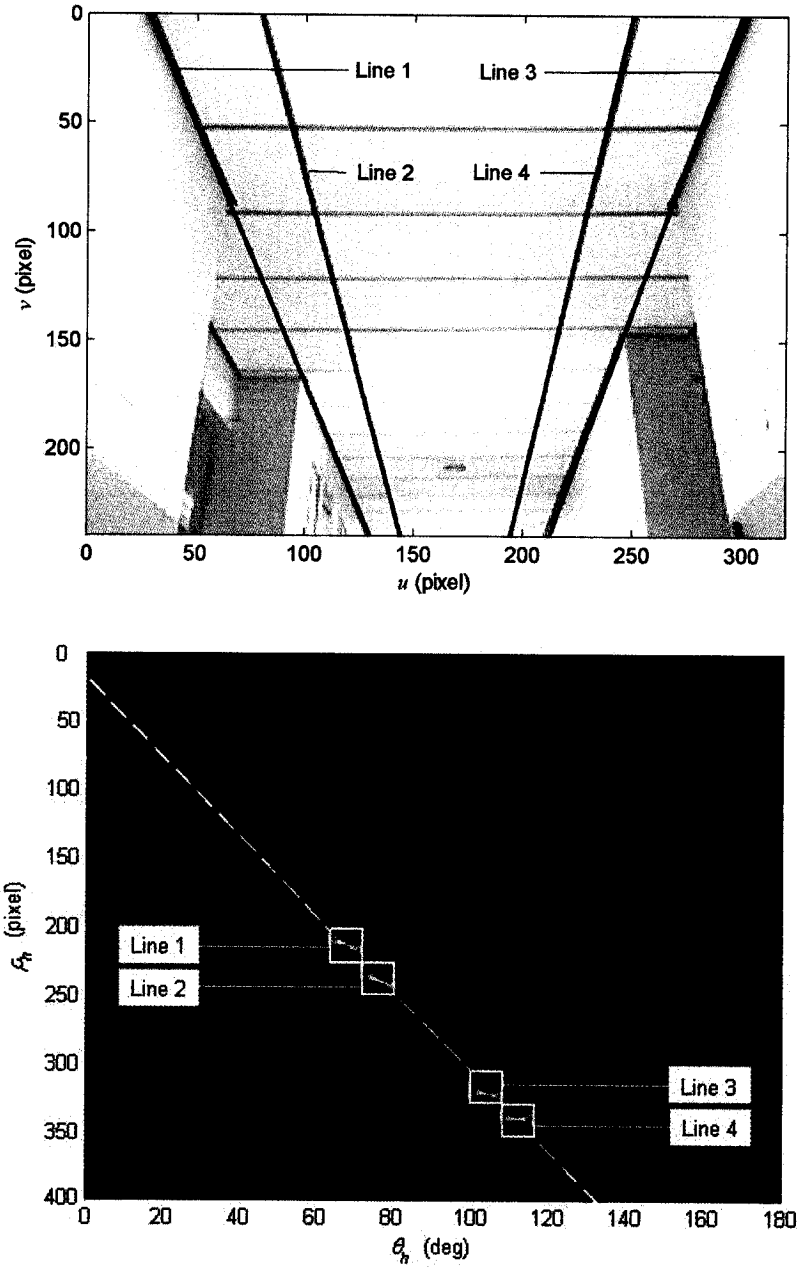


Figure 8.1: Particular pattern of peaks in the Hough space. (top) four lines intersecting at the vanishing point, and (bottom) the corresponding Hough space.

Figure 8.1(top) shows four line features, extracted from a ceiling scenery. The lines all intersect at a vanishing point situated outside the field of view. Shown superimposed over the original image, are the extracted lines features. The corresponding Hough space that results from the image transformation is shown in Figure 8.1(bottom).

The figure shows that the local maxima formed by the line passing by the vanishing point map in the Hough space are all along the same ridge line. Directly searching for maxima along this ridge line would offer three main advantages. First, it has the potential to eliminate altogether the extraction of line features of lesser interest with regards to the vanishing point analysis. Second, it is believed that this would make the image processing algorithm less sensitive to artefacts introduced due to varying scene illumination which could potentially solve the problems associated with the binarization thresholding factor. Finally, because the Hough transform is computationally demanding, searching only a region of interest of the Hough space would decrease the processing times, ultimately leading to a faster processing of the image, and faster updates of the pose tracking controller.

8.3.3 Large-Scale Navigation

Taken as a whole, the preliminary findings reported in this thesis have shown the feasibility of vision-based autonomous navigation in a small-scale environment. Experimental results indicate that the method is promising for real-world settings, particularly, because vanishing points are virtually present in every indoor environment.

Aside from the localization and control problems, practical autonomous navigation requires that the robot have the ability to perform higher deliberative tasks, to detect and avoid obstacles, and be able to plan its displacement. To achieve this, a number of enhancements to the current

vehicle architecture have been proposed. The successful implementation of both would open the doors to transferring the methods and techniques developed in this work to larger-scale environments allowing the elaboration of more complex cognitive tasks. Future work will be directed toward realizing this goal.

References

[1] UNECE and IFR, eds., *World Robotics 2005: Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment*, Geneva, Switzerland: United Nations, 2005.

[2] P.V.C. Hough, "Machine analysis of bubble chamber pictures," *International Conference on High Energy Accelerators and Instrumentation*, 1959, pp. 554–556.

[3] A. Aboshosha and A. Zell, "Disambiguating robot positioning using laser and geomagnetic signatures," *Intelligent Autonomous Systems (IAS-8), in Proceedings of the 8th Conf.*, Amsterdam, The Netherlands, March 10–13, 2004.

[4] J. Borenstein, H.R. Everett, and L. Feng, *Navigating Mobile Robots: Systems and Techniques*, Wellesley, MA.: A.K. Peters Ltd., 1996.

[5] S. Suksakulchai, S. Thongchai, D.M. Wilkes, and K. Kawamura "Mobile robot localization using an electronic compass for corridor environment," *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 5, 2000, pp. 3354–3359.

[6] V. Varveropoulos, "Robot localization and map construction using sonar data" [online], Dec. 2004 cited Feb. 16, 2006, available from the World Wide Web: <http://rossum.sourceforge.net>.

[7] B. Barshan and H.F. Durrant-Whyte "An inertial navigation system for a mobile robot," *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, vol. 3, 1993, pp. 2243–2248.

[8] L. Ojeda and J. Borenstein, "Experimental Results with the KVH C-100 fluxgate compass in mobile robots," *Proceedings of the IASTED International Conference on Robotics and Applications*, 2000, pp.156–162.

[9] B. Barshan and H. Durrant-Whyte, "Inertial navigation systems for mobile robots", *IEEE Trans. Robotics and Automation,* vol. 11, no. 3, June 1995, pp. 328-342.

[10] J.D. Tardos, J. Neira, P.M. Newman, and J.J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *International Journal of Robotics Research*, vol. 21, no. 4, 2002, pp. 311–330.

[11] J.J. Leonard and H.F. Durrant-Whyte. "Simultaneous map building and localization for an autonomous robot," *In IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS '91)*, pp. 1442–1447, Osaka, Japan, 1991.

[12] S. Walter, "The sonar ring: Obstacle detection for a mobile robot," *Robotics and Automation. Proceedings of the 1987 IEEE International Conference on*, vol. 4, 1987, pp. 1574–1579.

[13] R.G. Brown and B.R. Donald, "Mobile robot self-localization without explicit landmarks," *Algorithmica*, vol. 26, no. 3–4, Mar./Apr. 2000, pp. 515–559.

[14] S. Cooper and H. Durrant-Whyte, "A Kalman filter model for GPSnavigation of land vehicles," *in Proc. IEEE/RSJ/GI Int. Conf. Intelligent.Robot Systems*, Munich, Germany, Sept. 1994, pp. 157–163.

[15] H. Surmann, A. Nuchter, and J. Hertzberg. "An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, Dec. 2003, pp. 181–198.

[16] L. Kleeman, "Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning," *in Proc. 1992IEEE Int. Conf. Robot. Automat.*, Nice, France, 1992, pp. 2582–2587.

[17] A. Ohya, A.Kosaka, and A.C. Kak, "Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, 1998, pp. 969–978.

[18] D. Fernandez and A. Price, "Visual odometry for an outdoor mobile robot," *in Proceedings of the 2004 IEEE Conference on Robotics,Automation and Mechatronics*, 2004, pp. 816–821.

[19] D. Nister, O. Naroditsky, and J. Bergen, "Visual Odometry," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)* , vol. 1, 2004, pp. 652–659.

[20] Y. Ma, J. Kosecka, and S. Sastry, "Vision guided navigation for a nonholonomic mobile robot," *Robotics and Automation, IEEE Transactions on*, 1999, vol. 15, 521–536.

[21] R. Schuster, N. Ansari, and A. Bani-Hashemi "Steering a robot with vanishing points," *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 4, 1993, pp. 491–498.

[22] J. Heikkila and O. Silven, "A Four-step camera calibration procedure with implicit image correction," *Computer Vision and Pattern Recognition (CVPR '97), in Proceedings,* 1997, pp. 1106–1112.

[23] J.-Y. Bouguet, Camera Calibration Toolbox for Matlab, Public domain internet software, 1998.

[24] Pittman Corporation, "GM9236 Lo-Cog DC servo gearmotor," product datasheet.

[25] A.F. Winfield and O.E. Holland, "The application of wireless local area network technology to the control of mobile robots," *Microprocessors and Microsystems*, vol. 23, no. 10, 2000, pp. 597-607.

[26] Y. Zhao and S.L. BeMent, "Kinematics, dynamics and control of wheeled mobile robots," *Robotics and Automation, 1992. Proceedings of the 1992 IEEE International Conference on*, vol.1, 1992, pp. 91–96.

[27] J.C. Alexander and J.H. Maddocks, "On the kinematics of wheeled mobile robots," *International Journal Robotic Research*, vol. 8, no. 5, 1989, pp. 15–27.

[28] P. Muir and C. Neuman, "Kinematic modeling of wheeled mobile robots," *Journal of Robotic Systems*, vol. 4, no. 2, 1987, pp. 281–340.

[29] J.I. Neimark and N.A. Fufaev, *Dynamics of nonholonomic systems*, Providence, R.I.: American Mathematical Society, 1972.

[30] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of wheeled mobile robots: An experimental overview," *RAMSETTE: Articulated and Mobile Robots for SErvices and TEchnology*, S. Nicosia, B. Siciliano, A. Bicchi, and P. Valigi, Ed. London, U.K.: Springer-Verlag, 2001, vol. 270.

[31] G.F. Franklin, M.L. Workman, and D. Powell, *Digital Control of Dynamic Systems*, 3rd ed., Reading, MA.: Addison-Wesley, 1997.

[32] H. Olsson, K.J. Åstrom, C. Canudas de Wit, M. Gäfvert, and P. Lischinsky, "Friction Models and Friction Compensation," *European Journal of Control*, no.4, 1998, pp. 176–195

[33] A. O'Dwyer, *Handbook of PI and PID Controller Tuning Rules*, London.: Imperial College Press, 2003.

[34] H. Xu and S.X. Yang, "Tracking control of a mobile robot with kinematic and dynamic constraints," *Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on*, pp. 125–130, 2001.

[35] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer vision*, Cambridge : Cambridge University Press, 2003.

[36] D. Khadraoui, G. Motyl, P. Martinet, J. Gallice and F. Chaumette, "Visual servoing in robotics scheme using a camera/laser-stripe sensor," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 5, 1996, pp. 743–750.

[37] R. Berry and J. Burnell, *The Handbook of Astronomical Image Processing*, Richmond, VA.: Willmann-Bell, 2000.

[38] C.C. Slama, ed., *Manual of Photogrammetry*, 4th ed., Falls Church, VA.: American Society of Photogrammetry, 1980.

[39] R.G. Willson and S.A. Shafer, "What is the center of the image?," *in Proceedings of the 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '93)*, June, 1993, pp. 670–671.

[40] D.C. Brown "Decentering distortion of lenses," *Photogrammetric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[41] R. Kingslake, *Lenses in Photography: the practical guide to optics for photographers*, Rochester, NY.: Garden City Books, 1951.

[42] M. Young, *Optics and Lasers*, 3rd ed., Berlin : Springer-Verlag, 1986.

[43] Y. Le Grand, *Light, Color and Vision*, 2nd ed., London : Chapman and Hall Ltd., 1968.

[44] Sony Corporation, "ICX098AK - Diagonal 4.5mm (Type 1/4) Progressive Scan CCD Image Sensor with Square Pixel for Color Cameras," product datasheet.

[45] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 4, 1987, pp. 323–344.

[46] B.R. Dewey, *Computer Graphics for Engineers*, New York : Harper & Row, 1988.

[47] D.J. Kriegman and T.O. Binford, "Generic models for robot navigation," *Robotics and Automation, in Proceedings of the IEEE International Conference on*, vol.2, 1988, pp. 746–751.

[48] M.E. Mortenson, *Geometric transformations*, Industrial Press, 1995.

[49] A. Kelly, "Essential kinematics for autonomous vehicles," Report No. CMU-RI-TR-94-14, 1994.

[50] J. Guerrero and C. Sagüés, "Navigation from uncalibrated monocular vision," *3rd IFAC Conference on Intelligent Autonomous Vehicles*, 1998, pp. 210–215.

[51] C. De Boor, *A Practical Guide to Splines*, New York : Springer, 1978.

[52] S. Montiel and A. Ros, *Curves and Surfaces*, American Mathematical Society, 2005.

[53] Z. Fan, J. Borenstein, D. Wehe and Y. Koren, "Experimental evaluation of an Encoder Trailer for dead-reckoning in tracked mobile robots," *Intelligent Control, 1995., Proceedings of the 1995 IEEE International Symposium on*, 1995, pp. 571–576.

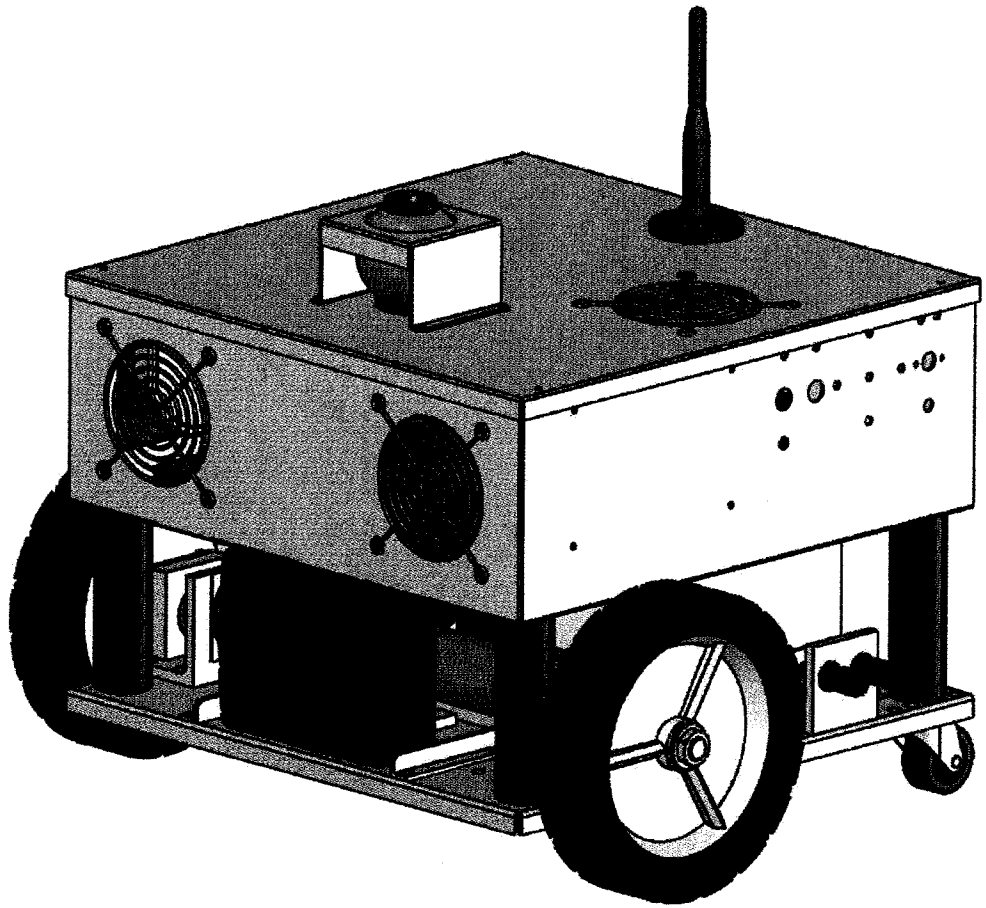
Appendix A

Mechanical Prototype

Overview

This appendix includes the preliminary assembly drawings of the robotic vehicle. Minor modifications to the original design have been made during the manufacturing but are not shown here.

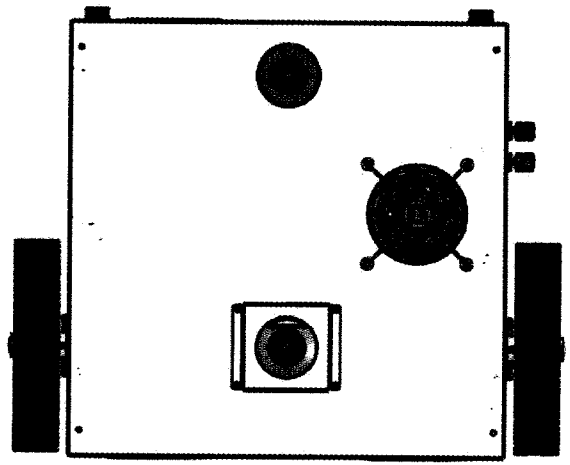
Drawings

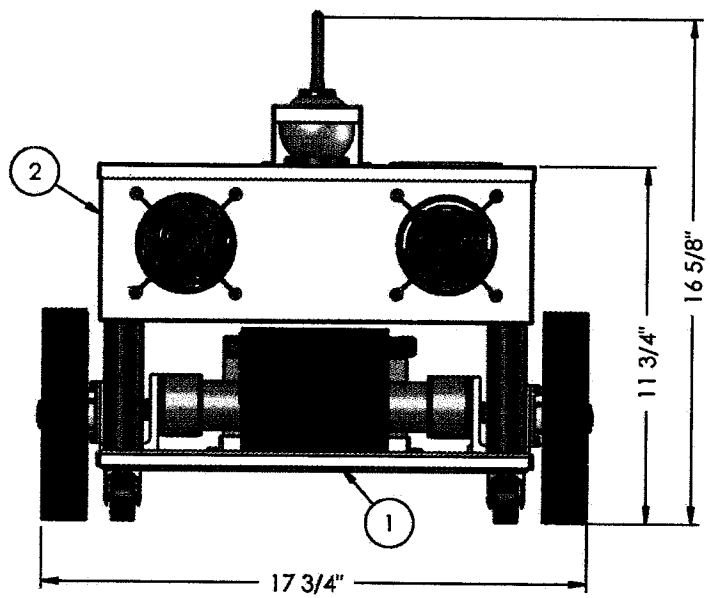


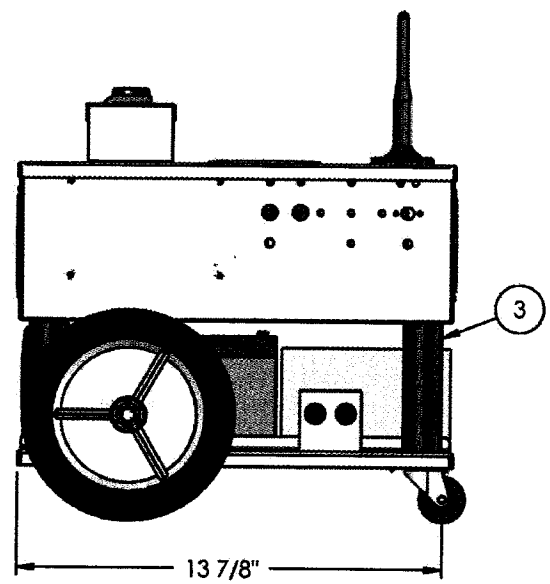
DIMENSIONS ARE IN INCHES		NAME	DATE	TITLE: ROBOT ASSEMBLY
TOLERANCES:	DRAWN	D.G.R.	10/09/2004	
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED			
	ENG APPR.			SIZE: A — DWG. NO.: 01-00-000 — REV: 0
	MFG APPR.			
MATERIAL:	QUANTITY: 1			SCALE: 1:4 — WEIGHT: — SHEET 1 OF 2

5 4 3 2 1

NO.	PART DESCRIPTION	REFERENCE	QTY.
1	LOWER DECK ASSEMBLY	01-01-000	1
2	UPPER DECK ASSEMBLY	01-02-000	1
3	CASE STANDOFF	01-00-001	4







186

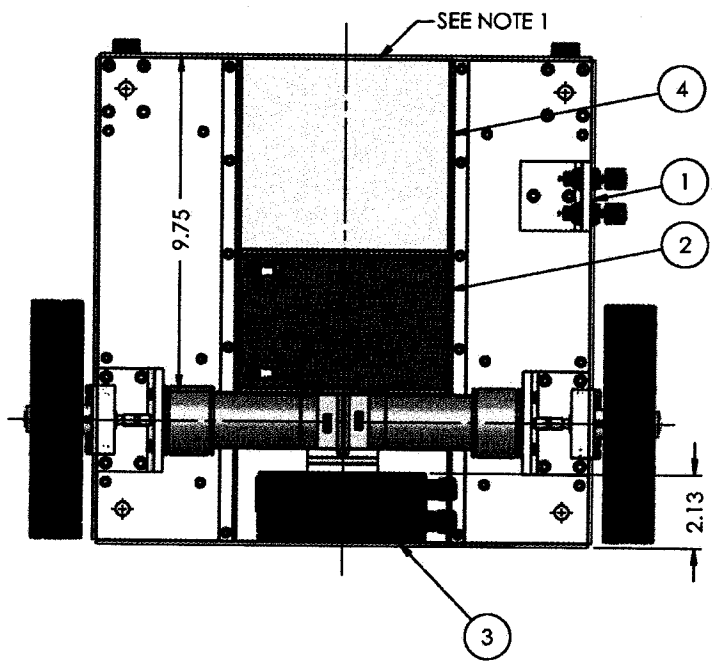
DIMENSIONS ARE IN INCHES		NAME	DATE
TOLERANCES:	DRAWN	D.G.R.	10/09/2004
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED		
	ENG APPR.		
	MFG APPR.		
MATERIAL:	QUANTITY: 1		

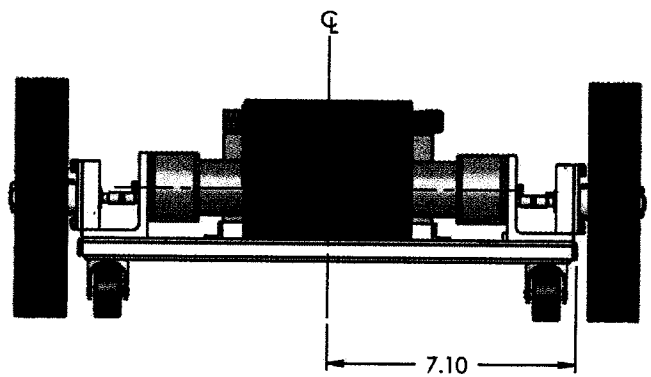
TITLE: ROBOT ASSEMBLY

SIZE	DWG. NO.	REV
A	01-00-000	0

SCALE: 1:6 | WEIGHT: | SHEET 2 OF 2

5 4 3 2 1





NO.	PART DESCRIPTION	REFERENCE	QTY.
1	PLATFORM ASSEMBLY	01-01-100	1
2	BATTERY	SEE SPECS	1
3	DC-AC CONVERTER	SEE SPECS	1
4	POWER SUPPLY	SEE SPECS	1
5	BINDING POST (BLACK)	SUPERIOR ELECTRIC, 5-WAY BINDING POST, BP30-2BR	1
6	BINDING POST (RED)	SUPERIOR ELECTRIC, 5-WAY BINDING POST, BP30-2BR	1

187

		NAME	DATE
DIMENSIONS ARE IN INCHES			
TOLERANCES:	DRAWN	D.G.R.	10/9/2004
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED		
	ENG APPR.		
	MFG APPR.		
MATERIAL: C1045	QUANTITY: 1		

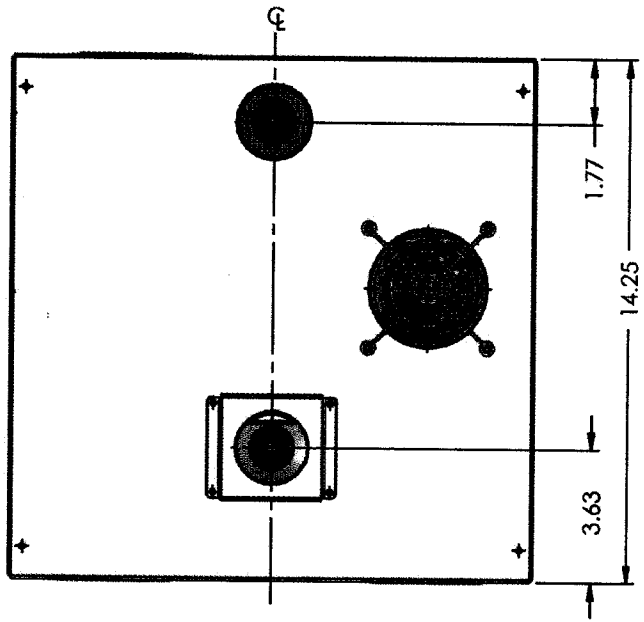
TITLE: LOWER DECK ASSEMBLY

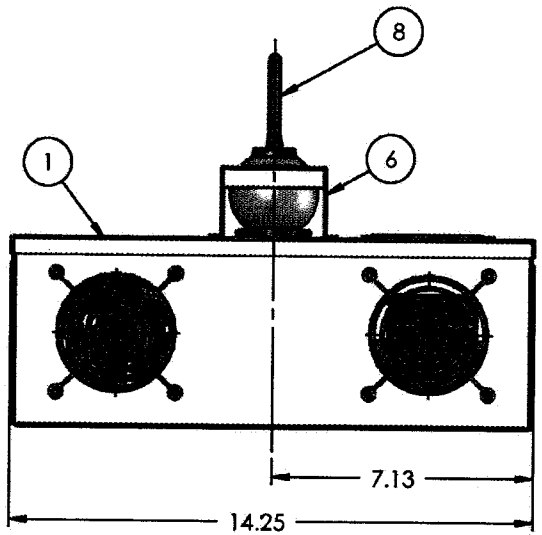
SIZE	DWG. NO.	REV
A	01-01-000	0

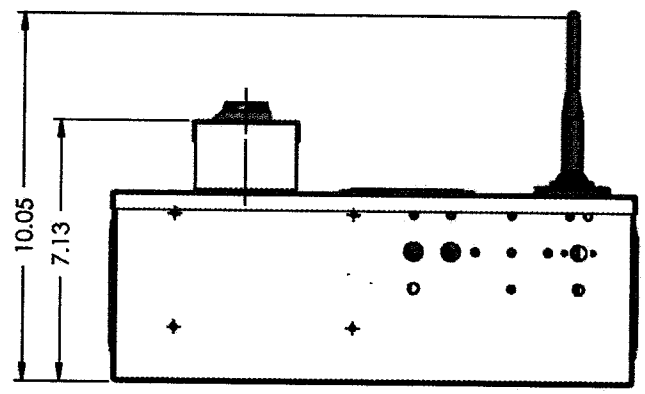
SCALE: 1:5 | WEIGHT: | SHEET 1 OF 1

5 4 3 2 1

NO.	PART DESCRIPTION	REFERENCE	QTY.
1	CASE ASSEMBLY	01-02-100	1
2	MOTHERBOARD ASSEMBLY	SEE SPECS.	1
3	HARDRIVE	SEE SPECS.	1
4	MICROCONTROLLER BOARD ASSEMBLY	SEE SPECS.	1
5	HARDWARE CONTROL BOARD ASSEMBLY	SEE SPECS.	1
6	CAMERA ASSEMBLY	01-02-200	1
7	CASE FAN	SEE SPECS.	1
8	ANTENNA	SEE SPECS.	1







188

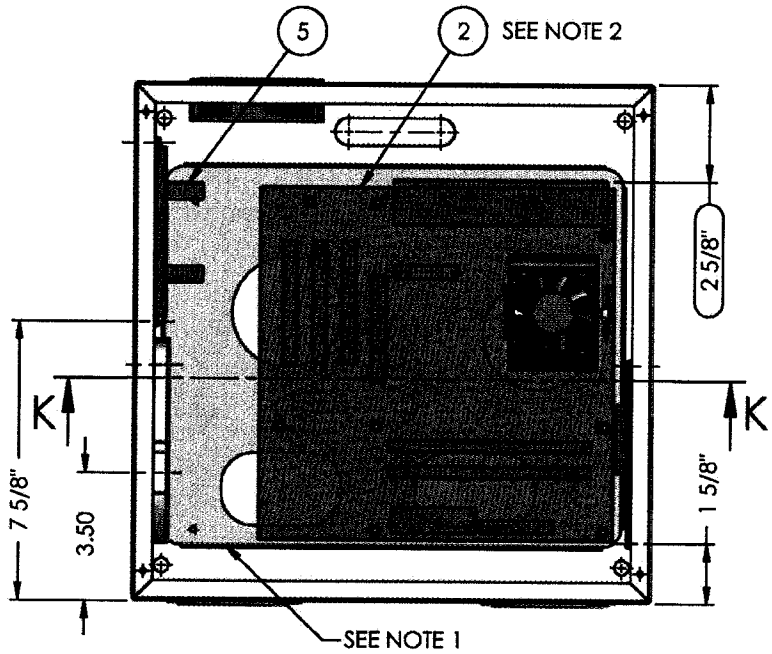
		NAME	DATE
DIMENSIONS ARE IN INCHES			
TOLERANCES:	DRAWN	D.G.R.	10/9/2004
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED		
	ENG APPR.		
	MFG APPR.		
MATERIAL: C1045	QUANTITY: 1		

TITLE:
UPPER DECK ASSEMBLY

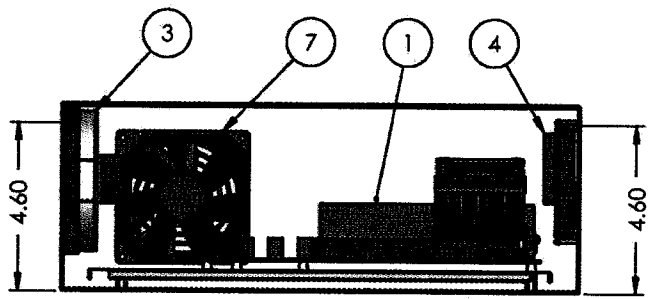
SIZE	DWG. NO.	REV
A	01-02-000	0

SCALE: 1:5 | WEIGHT: | SHEET 1 OF 2

5 4 3 2 1



189



SECTION K-K

DIMENSIONS ARE IN INCHES		NAME	DATE
TOLERANCES:	DRAWN	D.G.R.	10/9/2004
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED		
	ENG APPR.		
	MFG APPR.		
MATERIAL: C1045	QUANTITY: 1		

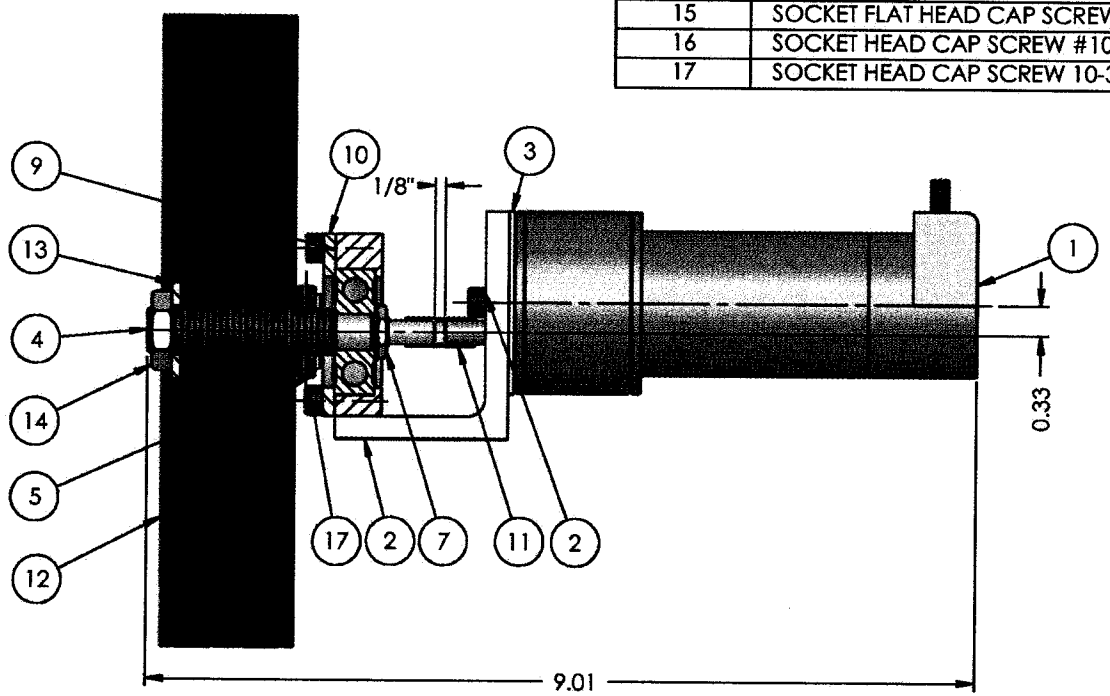
TITLE: UPPER DECK ASSEMBLY

SIZE	DWG. NO.	REV
A	01-02-000	0

SCALE: 1:5 | WEIGHT: | SHEET 2 OF 2

5 4 3 2 1

NO.	PART DESCRIPTION	REFERENCE	QTY.
1	MOTOR	PITTMAN GM9236 (SEE SPECS.)	1
2	MOTOR SUPPORT	01-01-301	1
3	MOTOR SHIM	01-01-302	1
4	SHAFT	01-01-303	1
5	ROLL PIN 0.156 x 1	STEEL	1
6	DEEP GROOVE BALL BEARING	SKF 6300-2RSH	1
7	HEX JAM NUT 0.375-24 UNC	STEEL	1
8	BEARING WASHER	STEEL	1
9	HOUSING	01-01-304	1
10	HOUSING COVER	01-01-305	1
11	COUPLING	01-01-306	1
12	WHEEL	01-01-307	1
13	PLAIN WASHER 0.500	STEEL	1
14	HEX JAM NUT 0.5-20 UNC	STEEL	1
15	SOCKET FLAT HEAD CAP SCREW #10-32 x 0.75	STEEL	2
16	SOCKET HEAD CAP SCREW #10-32 x 1	STEEL	2
17	SOCKET HEAD CAP SCREW 10-32 x 0.375	STEEL	4



DIMENSIONS ARE IN INCHES		NAME	DATE
TOLERANCES:	DRAWN	D.G.R.	10/09/2004
FRACTIONAL ± 0.016 TWO PLACE DECIMAL ± 0.010 THREE PLACE DECIMAL ± 0.001 ANGULAR: ± 1°	CHECKED		
	ENG APPR.		
	MFG APPR.		
MATERIAL:	QUANTITY: 2		

TITLE: MOTOR ASSEMBLY

SIZE	DWG. NO.	REV
A	01-01-300	0

SCALE: 1:2 | WEIGHT: | SHEET 1 OF 1

5 4 3 2 1

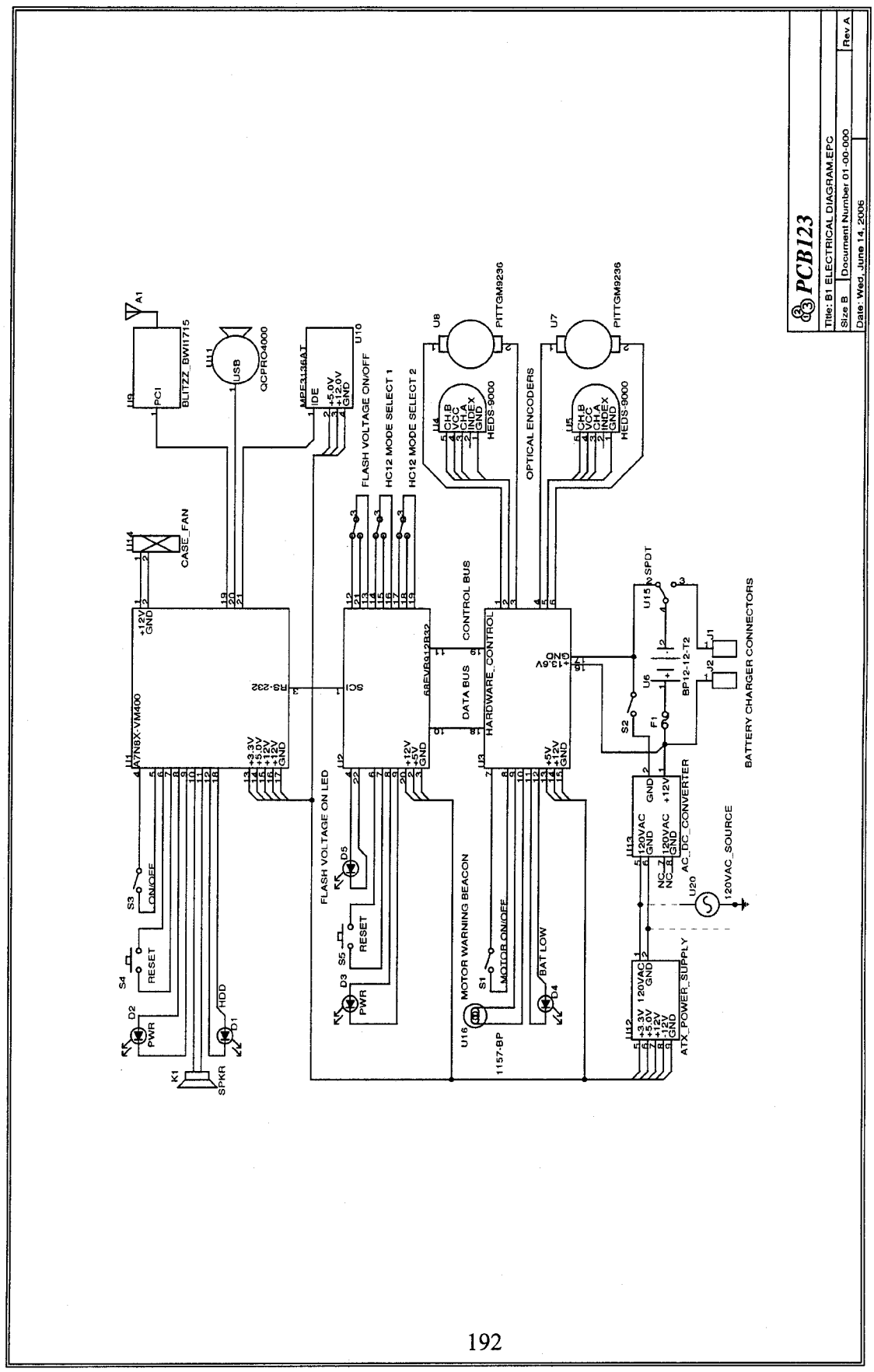
Appendix B

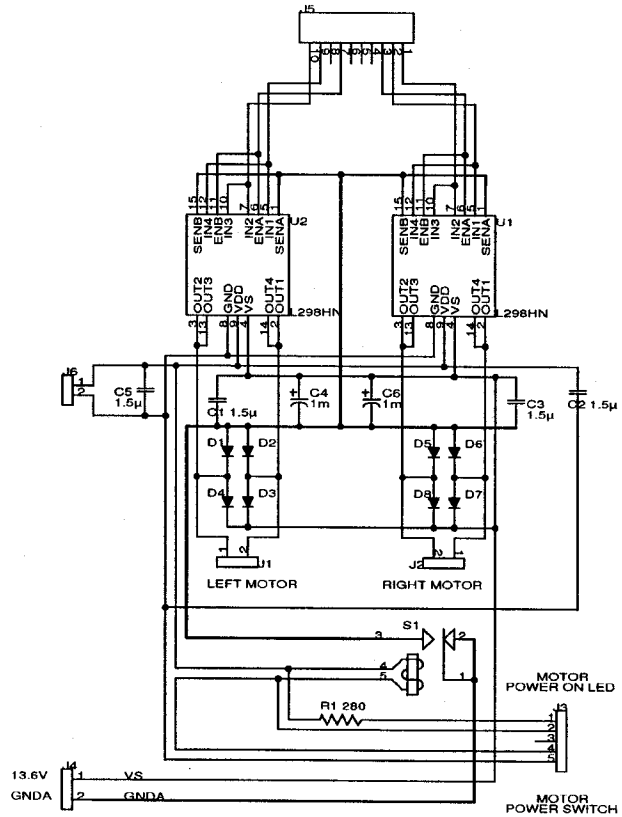
Electrical System

Overview

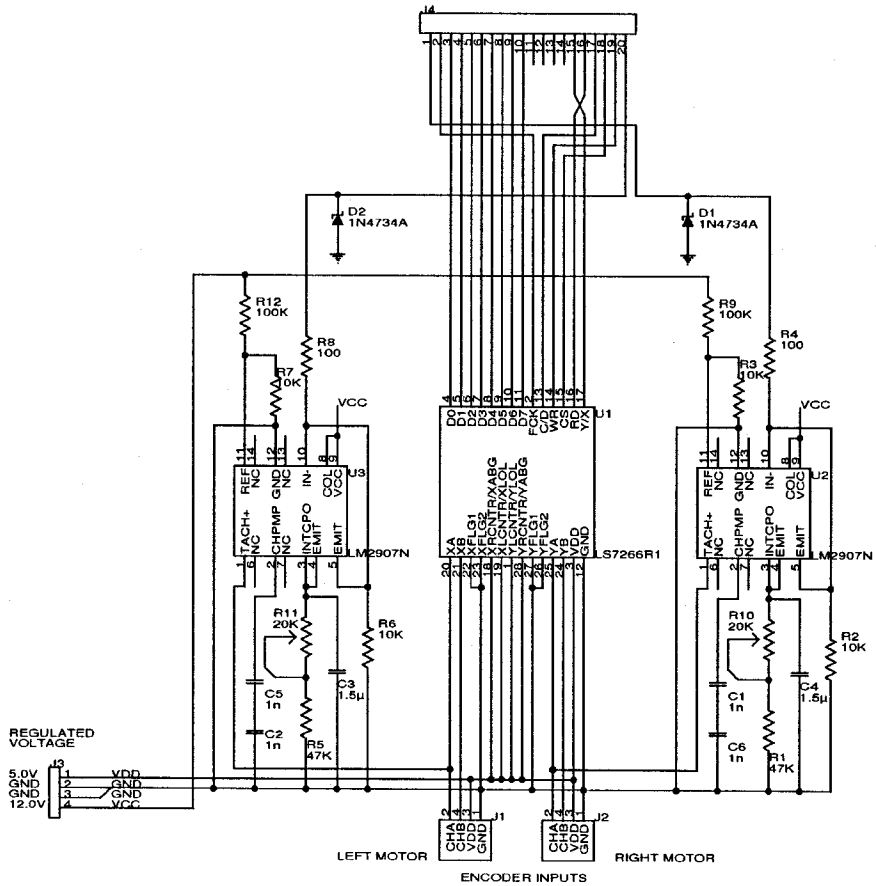
This appendix includes the electrical diagrams of the robotic vehicle and of the custom-engineered electronic circuitry. Schematics of the motor drivers and sensory boards are also included along with their respective printed circuit board layouts.

Drawings


PCB123
Title: B1 ELECTRICAL DIAGRAM.EPC
Size: B
Document Number: 01-00-000
Rev: A
Date: Wed, June 14, 2006
A1
U9
PCI
BLIT2Z_BWII715
U11
USB
QCPRO4000
U10
MPF31136AT
IDE
+5.0V
+12V
GND
FLASH VOLTAGE ON/OFF
HC12 MODE SELECT 1
HC12 MODE SELECT 2
U8
PITTGM9236
U4
CH.B
CH.C
CH.A
INDEX
GND
HEDS-9000
OPTICAL ENCODERS
U7
PITTGM9236
U5
HEDS-9000
U14
CASE_FAN
+12V
GND
U1
A7N8X-VM400
+3.3V
+5.0V
+12V
GND
RS-232
SCI
U2
68EVB912B32
CONTROL BUS
DATA BUS
U3
HARDWARE_CONTROL
+13.6V
NO
+5V
+12V
GND
U15
N.SPDT
U6
BP12-12-T2
F1
S2
J1
J2
BATTERY CHARGER CONNECTORS
FLASH VOLTAGE ON LED
D5
S5
RESET
D3
PWR
S3
ON/OFF
S4
RESET
D2
PWR
HDD
D1
K1
SPKR
U16
1157-BP
MOTOR WARNING BEACON
S1
MOTOR ON/OFF
BAT LOW
D4
U13
120VAC
GND
NC_7
NC_8
120VAC
+12V
GND
AC_DC_CONVERTER
U20
120VAC_SOURCE
U12
+3.3V
120VAC
+5.0V
+12V
GND
ATX_POWER_SUPPLY

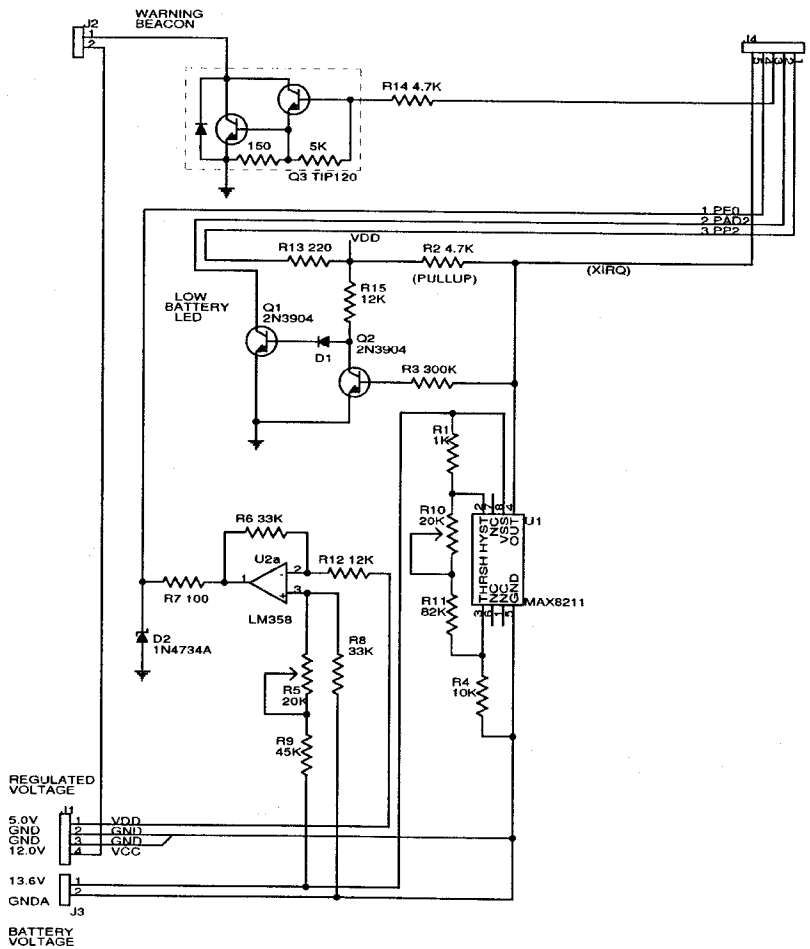


PCB123		
Title: B2 MOTOR DRIVER BOARD.EPC		
Size B	Document Number	Rev A
Date: Wed, June 14, 2006		



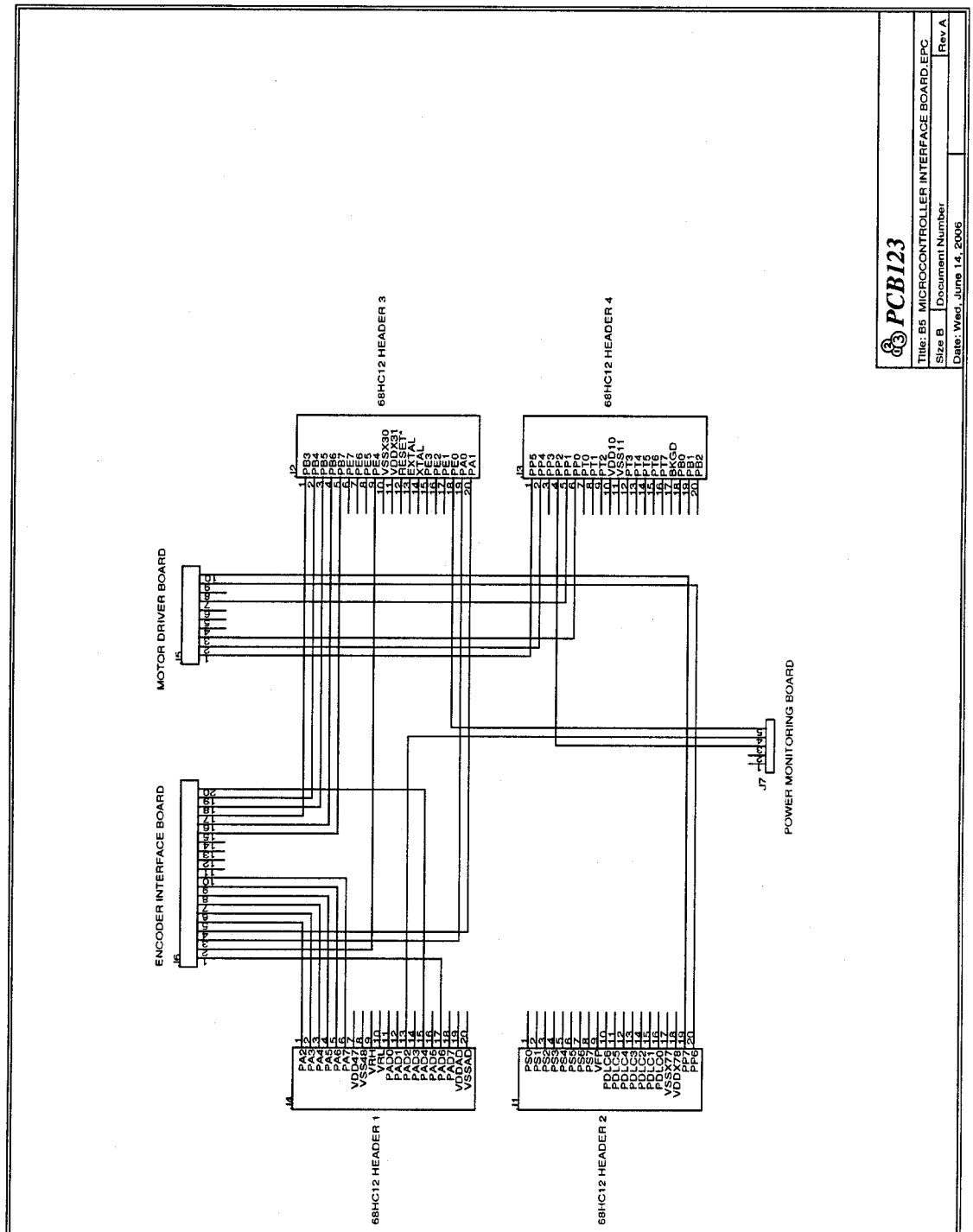
PCB123

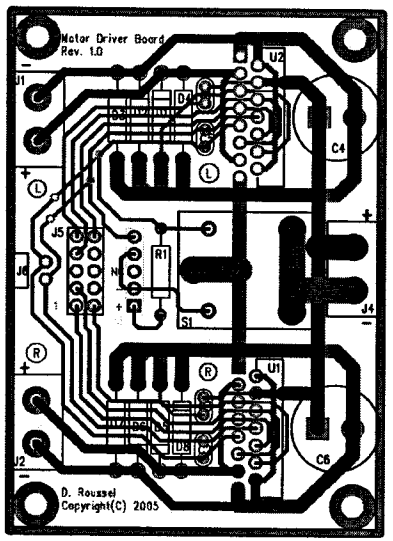
Title: B3 POSITION - VELOCITY SENSOR BOARD.EPC		
Size B	Document Number	Rev A
Date: Wed, June 14, 2006		



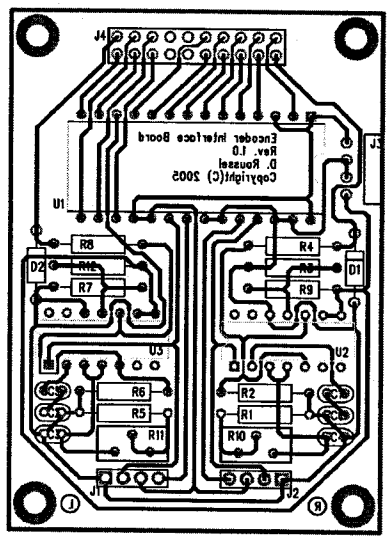
PCB123

Title: B4 POWER MONITORING BOARD.EPC		
Size B	Document Number	Rev T
Date: Wed, June 14, 2006		

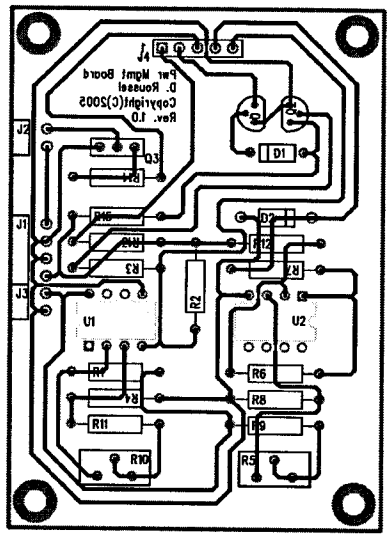

PCB123
Title: B5 MICROCONTROLLER INTERFACE BOARD.EPC
Size: B
Document Number
Rev: A
Date: Wed, June 14, 2006
68HC12 HEADER 3
68HC12 HEADER 4
MOTOR DRIVER BOARD
ENCODER INTERFACE BOARD
POWER MONITORING BOARD
68HC12 HEADER 1
68HC12 HEADER 2



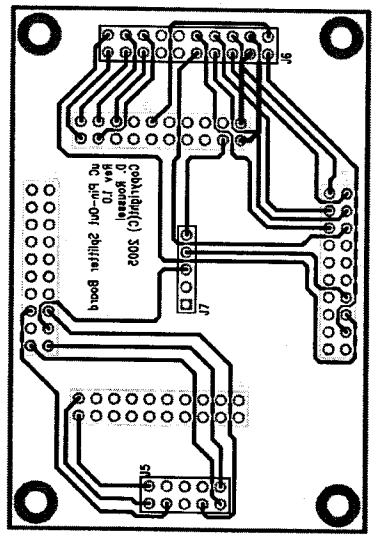
MOTOR DRIVER BOARD



POSITION/VELOCITY SENSOR BOARD



POWER MONITORING BOARD



MICROCONTROLLER INTERFACE BOARD

PCB123		
Title: B6 PCB LAYOUTS.EPC		
Size: B	Document Number	Rev: A
Date: Wed, June 14, 2006		

Appendix C

Source Code Listings

Overview

This appendix includes the computer programs developed within the framework of this research. It includes the low-level embedded firmware programmed in assembly language using the *CPU12* instruction set, the Human-Machine Interface code developed with *Visual C++ .net 2003*, and the high-level navigational algorithm developed using *Maltlab® version 7.0.1 (R14) SP1* and compiled as a stand-alone *C++* dynamic library (DLL) using *Matlab® Compiler 4.0.*

Listings

C.1 Low-Level Embedded Firmware

```
                NOLIST
;---------------------------------------------------------------------------------------------
;File name:             UserEquates.inc
;
;Description:           Register and bit definitions.
;                       Adapted from: Doughman, Programming the Motorola M68Hc12 Family, 2000.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;---------------------------------------------------------------------------------------------
;          Common address definitions
                INCLUDE "config.inc"
                INCLUDE "limregs.inc"
                INCLUDE "clocksyn.inc"
                INCLUDE "eeprom.inc"
                INCLUDE "flashee.inc"
                INCLUDE "msi.inc"
                INCLUDE "pwm.inc"
                INCLUDE "a2d.inc"
                INCLUDE "timer.inc"
                INCLUDE "ascii.inc"

                ;Memory Mapping Equates for D-Bug12 mode (limited RAM)
                IFDEF   DBUG12
RAM_START       equ     RAMBase
RAM_SIZE        equ     RAMSize

ROM_START       equ     EEBase
ROM_SIZE        equ     EESize

STACK_HI        equ     $0A00
STACK_SIZE      equ     $0100                   ;256 bytes of RAM memory
STACK_LO        equ     STACK_HI-STACK_SIZE

INT_VEC_JMP_TBL equ     $0B00
                ENDIF

                ;Memory Mapping Equates for BDM Probe Mode (no write access to flash EEPROM)
                IFDEF   POD_DBUG12
RAM_START       equ     RAMBase
RAM_SIZE        equ     RAMSize

ROM_START       equ     EEBase
ROM_SIZE        equ     EESize

STACK_HI        equ     RAM_START+RAM_SIZE
STACK_SIZE      equ     $0100                   ;256 bytes of RAM memory
STACK_LO        equ     STACK_HI-STACK_SIZE

INT_VEC_JMP_TBL equ     $0B00
                ENDIF

                ;User application (write access to flash EEPROM)
                IFDEF   POD_USERAPP
RAM_START       equ     RAMBase
RAM_SIZE        equ     RAMSize

ROM_START       equ     EEBase
ROM_SIZE        equ     EESize

FLASH_START     equ     FEEBase
FLASH_SIZE      equ     FEESize

STACK_HI        equ     RAM_START+RAM_SIZE
STACK_SIZE      equ     $0100                   ;256 bytes of RAM memory
STACK_LO        equ     STACK_HI-STACK_SIZE

INT_VEC_JMP_TBL equ     $F7C0
                ENDIF

                ;Bitwise Equates
NONE            equ     %00000000
BIT0            equ     %00000001
BIT1            equ     %00000010
BIT2            equ     %00000100
BIT3            equ     %00001000
BIT4            equ     %00010000
BIT5            equ     %00100000
BIT6            equ     %01000000
BIT7            equ     %10000000
ALL             equ     %11111111

                ;Battery Voltage Level Equates
EMPTY           equ     %10000000               ;Byte Mask

                ;Speed Controller Equates
MAX_POS         equ     $7FFF                   ;Max speed forward constant
MAX_NEG         equ     $8001                   ;Max speed reverse constant
PP4             equ     BIT4                    ;Direction right axis
PP5             equ     BIT5                    ;Direction right axis
PP6             equ     BIT6                    ;Direction left axis
PP7             equ     BIT7                    ;Direction left axis
DIR             equ     BIT5
ATDCH_X         equ     BIT6|BIT2               ;Left motor ATD channel 4
ATDCH_Y         equ     BIT6|BIT2|BIT1          ;Right motor ATD channel 6
ATDCH_B         equ     BIT6|BIT1               ;Battery voltage monitoring ATD ch. 2

                ;LS7266R1 Encoder Counter Equates
SCALE           equ     $09                     ;Prescalar factor
RLD             equ     NONE                    ;Encoder IC reset/load signal decoder
CMR             equ     BIT5                    ;Encoder IC counter mode register
IOR             equ     BIT6                    ;Encoder IC I/O control register
IDR             equ     BIT6|BIT5               ;Encoder IC index control register
YX              equ     BIT7                    ;Encoder IC Channel X-Low / Y-High
RD              equ     BIT6                    ;Encoder IC Read-Low
CS              equ     BIT5                    ;Encoder IC chip-select (low)
WR              equ     BIT4                    ;Encoder IC Write-Low
```

```
CD              equ     BIT3                    ;Encoder IC Control-High / Data-Low

                ;SCI Equates
RXBUFSIZE       equ     $20                     ;32 bytes of receiver buffer size
TXBUFSIZE       equ     $20                     ;32 bytes of transmitter buffer size
BAUDRATE        equ     B9600                   ;Serial communication baud rate

                NOLIST
;-------------------------------------------------------------------------------
;File name:             CONSTANT.inc
;
;Description:           Definition of the constants.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             MC68HC912B32
;-------------------------------------------------------------------------------
                LIST

                ;Constants
PERIOD_M        dc.b    127                     ;Motors PWM period
PERIOD_B        dc.b    255                     ;Beacon light period

               ;SCI Message Identifier
MsgSpeedRef     dc.b    'S'                     ;SCI Message identifier
MsgPIDGain      dc.b    'G'                     ;SCI Message identifier
MsgMotorEnable  dc.b    'E'                     ;SCI Message identifier
SCIMsgTableSize equ     *

                ;Interrupt Messages
HANDLERStrg     dc.b    "HANDLER.",CR,LF,NUL
RTIStrg         dc.b    "RTI.",CR,LF,NUL
PWRFULLStrg     dc.b    "Battery fully charged.",CR,LF,NUL
PWRHALFStrg     dc.b    "Battery power OK.",CR,LF,NUL
PWRLOStrg       dc.b    "Battery power low.",CR,LF,NUL
PWREMPTYStrg    dc.b    "Battery discharged.",CR,LF,NUL
                SWITCH  sRAMVAR

                ;Motor Data Structure
                OFFSET  0
AtdCh           ds.b    1
PwdtyPtr        ds.b    2
Polarity1       ds.b    1
Polarity2       ds.b    1
EncStat         ds.b    1
AxisStat        ds.b    1
Position        ds.b    4                       ;Motor inc position reading
Speed           ds.b    2                       ;Motor speed
SpeedRef        ds.b    2                       ;Motor speed set point
Uk              ds.b    4                       ;Motor current control input
Ek              ds.b    2                       ;Motor current error
Duty            ds.b    2                       ;Motor duty cycle
                ds.b    9                       ;Buffer to align memory rows (easier to read)
StructSize      equ     *

                SWITCH  sRAMVAR
XAxisStructPtr  ds.b    StructSize
YAxisStructPtr  ds.b    StructSize
MainLoopCounter ds.b    1                       ;Counter

K               ds.w    1                       ;Integral gain
Kc              ds.w    1                       ;Proportional gain
BatStat         ds.b    1                       ;Battery status register
                ds.b    10                      ;Buffer to align memory rows (easier to read)

                ;Serial Communication
RxBuffPtr       ds.b    RXBUFSIZE               ;Pointer to the Receiver queue.
TxBuffPtr       ds.b    TXBUFSIZE               ;Pointer to the Transmitter queue size.
RxIn            ds.b    1                       ;Next available location in the Rx queue.
RxOut           ds.b    1                      ;Next character to be removed from the Rx queue.
TxIn            ds.b    1                       ;Next available location in the Tx queue
TxOut           ds.b    1                       ;Next character to be sent from the Tx queue.
RxBAvail        ds.b    1                       ;Number of bytes left in the Rx queue.
TxBAvail        ds.b    1                       ;Number of bytes left in the Tx queue.


                NOLIST
;-------------------------------------------------------------------------------
;File name:     START_UP.asm
;
;Description:   1. Motor control
;               2. Speed feedback
;               3. Battery voltage level monitoring
;               4. Serial communication
;               4. Beacon light activation
;               5. Variable naming convention
;                       a. All caps = Constants or equates
;                       b. tXXXX = test variables
;                       c. lXXXX = Local variables
;                       d.  XXXX = global variables
;                       e. pXXXX = parameter of a routine
;                       f. rXXXX = return value of a routine
;
;Author:        D. Gagné-Roussel, University of Ottawa
;
;Date:          January 2005
;
;Processor:     Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST

                ;Assembler instructions
;DBUG12         equ     0                       ;D-Bug12 monitor
POD_USERAPP     equ     1                       ;BDM Probe interface mode (with flash)
;POD_DBUG12     equ     1                       ;BDM Probe interface mode (no flash)
tGETPOS         equ     0
MINIIDE         equ     1

                ;System equates and symbol definitions
                NOLIST                          ;Turn off listing generation
                INCLUDE "UserEquates.inc"       ;System address definitions for HC12
                INCLUDE "macros.inc"            ;Macro definition template
                LIST                            ;Turn on listing generation
                MLIST   ON                      ;Turn on listing generation for macros

                IF      MINIIDE==0
                XDEF    Entry                   ;Export 'Entry' symbol for abs assembly
                ENDIF

                ;Start of CODE Section
```

```
sCODE           SECTION
                IF      MINIIDE==1
                ORG     $8000
                ENDIF

Entry:

                INCLUDE "INIT.asm"              ;Do peripheral initializations
                jsr      MAIN                   ;Jump to the main function

                ;Include files
                INCLUDE "MAIN.asm"

                ;Routines
                INCLUDE "GETPOS.asm"
                INCLUDE "GETDIR.asm"
                INCLUDE "GETSPEED.asm"
                INCLUDE "GETPWRLEVEL.asm"
                INCLUDE "PID.asm"
                INCLUDE "CONTROL_INPUT.asm"
                INCLUDE "SETSPEEDREF.asm"
                INCLUDE "SETMOTORENABLE.asm"
                INCLUDE "SETPIDGAIN.asm"
                INCLUDE "SENDPWRLEVEL.asm"
                INCLUDE "SENDACTUALSPEED.asm"
                INCLUDE "PUTSTRG.asm"
                INCLUDE "GETCHAR.asm"
                INCLUDE "PUTCHAR.asm"
                INCLUDE "ATOHB.asm"
                INCLUDE "ATOHW.asm"
                INCLUDE "ADD32.asm"

                ;Interrupt service routines
                INCLUDE "SCI_ISR.asm"
                INCLUDE "PWR_LO_ISR.asm"
                INCLUDE "ISR_HANDLER.asm"
                INCLUDE "RTI_ISR.asm"

                ;Constants
                INCLUDE "CONSTANT.inc"

                ;Variables
sRAMVAR         SECTION
                IF      MINIIDE==1
                ORG     $0800
                ENDIF
                INCLUDE "DATA.inc"

                ;Interrupt vector jump table
                ORG     INT_VEC_JMP_TBL
                INCLUDE "INTVECTORTBL.inc"

                ;End of compilation
                END
```

```
                NOLIST
;---------------------------------------------------------------------------------------------
;File name:             INIT.asm
;
;Description:           MC68HC912B32 Initialization procedures.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;---------------------------------------------------------------------------------------------
                LIST

LIM_INIT:       lds     #STACK_HI               ;Memory address of stack upper boundary
                leax    0,SP                    ;Initialize stack frame pointer

                ldaa    #RegBase>>8             ;Get upper 8-bits of the reg base address
                staa    INITRG                  ;Move and lock the reg base address
                nop                             ;Required time of moving

                ldaa    #RAMBase>>8             ;Get the upper 8-bits of the RAM base address
                staa    INITRM                  ;Move and lock the RAM base address
                nop                             ;Required time of moving

                ldaa    #((EEBase>>8)&$f0)+EEON ;Get upper 4-bits of EEbase - EE on

                staa    INITEE                  ;Move and lock the EE base address
                nop                             ;Required time of moving

                ldaa    #ROMON|MAPROM           ;Make sure the flash remains on at $8000
                staa    MISC                    ;

                clr     COPCTL                  ;Disable the COP and clock monitor

                ldaa    #SMODN|ESTR|IVIS        ;Normal single chip mode, internal visibility
                staa    MODE                    ;Lock the operating mode

                ldaa    #NDBE                   ;PE4 is the external E-clock pin
                staa    PEAR                    ;Lock the external E-clock bit

                clr     RDRIV                   ;Lock PORT A,B,E normal pin drive levels

                ldy     #RAM_START              ;Clear ram memory of undefined values
                mSETRAM Y,$A0,$00

                ldy     #STACK_LO               ;Enable stack footprint $55
                mSETRAM Y,STACK_SIZE, $33

                ;IRQ interrupt
IRQ_INIT:       bset    INTCR,IRQEN             ;Local enable IRQ interrupts

                ;Real-time interrupt
RTI_INIT:       ldaa    #RSBCK|RTR2|RTR1|RTR0
                staa    RTICTL                  ;Enable RTI in Bkgd mode
                bset    RTICTL,RTIE             ;Local enable RTI interrupts

                ;Serial communication
SCI_INIT:       bclr    SC0CR1,LOOPS            ;Operate in normal mode
                movw    #BAUDRATE,SC0BDH        ;Initialize the baud rate register.
                clr     RxIn                    ;Set Rx queue indexes to 0.
```

```
                clr     RxOut
                clr     TxIn                    ;Set Tx queue indexes to 0.
                clr     TxOut
                ldab    #RXBUFSIZE              ;Init number of available Rx
                stab    RxBAvail                ;Queue bytes to RxBuffSize. Queue is empty.
                ldab    #TXBUFSIZE              ;Init number of available Tx
                stab    TxBAvail                ;Queue bytes to TxBuffSize. Queue is empty.
                ldab    #TE|RE|RIE              ;Get bit mask for Tx, Rx & Rx interrupt.
                stab    SC0CR2                  ;Enable Tx & Rx & Rx interrupts.

                ;Analog-to-digital convertion module
ATD_INIT:       movb    #%10000000,ATDCTL2      ;Enables the ATD
                mDELAY  $FF                     ;Power up required delay
                movb    #%00000000,ATDCTL3      ;Ignore freeze in ATDCTL3
                movb    #%00000011,ATDCTL4      ;Final sample time 8 ATD clocks

PORTP_INIT:     movb    #BIT7|BIT6|BIT5|BIT4,DDRP;Set PORTP pin 4,5,6,7 as output
                clr     PORTP

                ;Pulse width modulation
PWM_INIT:       movb    #%00010111,PWCLK        ;Sets channel 0, 1,2 and 3 as 8-bit channels
                                                ;Sets prescale factor of clock A = 4
                movb    #%01001111,PWPOL        ;Sets the polarity of the waveform
                                               ;Select clock A for ch 0 & 1, clock S1 for ch 2
                movb    #74,PWSCAL1             ;Clock S1 divided
               movb    #%00000000,PWCTL        ;Sets left-aligned waveform type for all channel

                movb    PERIOD_M,PWPER0         ;Set right motor period
                movb    PERIOD_M,PWPER1         ;Set left motor period
                movb    PERIOD_B,PWPER2         ;Set beacon light period
                movb    #0,PWDTY0               ;Initialize right motor duty cycle to 0%
                movb    #0,PWDTY1               ;Initialize left motor duty cycle to 0%
                movb    #42,PWDTY2              ;Initialize duty for beacon light
                bclr    PWEN,PWEN2|PWEN1|PWEN0  ;Disable PWM

ENCODER_INIT:
                ;XA-XB channels = Left motor encoder
                ;YA-YB channels = Right motor encoder
                ;PORTA = Data bus        | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
                ;                       | a7| a6| a5| a4| a3| a2| a1| a0|
                ;PORTB = Control bus     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
                ;                       |Y/X| RD| CS| WR|C/D| - | - | - |
                ;Default state          |X=0| 1 | 0 | 1 |C=1|n/a|n/a|n/a|

                ;Initialization of the Data and Control bus
                movb    #ALL,DDRA               ;Set PORTB as an output port
                movb    #ALL,DDRB               ;Set PORTA as an output port
                movb    #NONE,PORTB             ;Reset PORTB
                bset    PORTB,RD|WR|CD          ;Initialize PORTB, Ctrl mode (default)

                ;Set the counter mode
                clr     PORTA                   ;Reset PORTA
               bset    PORTA,BIT7|CMR|BIT3     ;Write CMR reg, X & Y,X1 mode, bin. count,normal
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Reset byte pointer
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|RLD|BIT0     ;Write RLD register, X & Y byte pointer reset
                bclr    PORTB,WR                ;Pulse the WR pin Low
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Load XPR0 with filter clock prescalers for X channel
                movb    #9,PORTA                ;PSC value of 9 to PR bit-0
                bclr    PORTB,CD                ;Data mode enable
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High
                bset    PORTB,CD                ;Control mode enable (Default)

                ;Load YPR0 with filter clock prescalers for Y channel
                movb    #9,PORTA                ;PSC value of 9 to PR bit-0
                bset    PORTB,YX                ;Select Y axis
                bclr    PORTB,CD                ;Data mode enable
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High
                bset    PORTB,CD                ;Control mode enable (Default)
                bclr    PORTB,YX                ;Bring YX bit to X=0 (Default)

                ;Transfering both filter clock X and Y PR0 to PSC
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|RLD|BIT4|BIT3;Transfer PR0 to PSC X and Y
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Reseting X and Y counters
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|RLD|BIT1     ;Write RLD register, reset X & Y CNTR
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Resetting flag register bits
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|RLD|BIT2     ;Write RLD register, reset X & Y BT, CT, CPT, S
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|RLD|BIT2|BIT1;Write RLD register, reset X & Y E bits
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Disabling index
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|IDR          ;Disable index
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Enabling A and B inputs
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,BIT7|IOR|BIT0     ;Write IOR register, enable A & B,
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             MAIN.asm
;
;Description:           Main program code.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
MAIN:

;Allocate stack frame memory
                OFFSET  0
lAxStructPtr    ds.b    2                       ;Local variables
lAxis           ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
;pParam1        ds.b    1                       ;Parameters
;pParam2        ds.b    1
;rVar1          ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values
                clr     MainLoopCounter         ;First RTI is responsible to initiate
                                                ;main routine looping
;Subroutine code
                ldy     #XAxisStructPtr         ;Peripheral assignment to X axis
                movb    #ATDCH_X,AtdCh  ,Y      ;ATD channel settings AN4
                movw    #PWDTY1,PwdtyPtr,Y      ;PWM Channel 1 Duty Cycle Register
                movb    #PP7,Polarity1  ,Y      ;Polarity of X axis driven with PP6
                movb    #PP6,Polarity2  ,Y      ;and PP7

                ldy     #YAxisStructPtr         ;Peripheral assignment to Y axis
                movb    #ATDCH_Y,AtdCh  ,Y      ;ATD channel settings AN6
                movw    #PWDTY0,PwdtyPtr,Y      ;PWM Channel 0 Duty Cycle Register
                movb    #PP4,Polarity1  ,Y      ;Polarity of X axis driven with PP4
                movb    #PP5,Polarity2  ,Y      ;and PP5

                ;PID tuning gains
                movw    #10,K                   ;Integral gain
                movw    #25,Kc                  ;Proportional gain

                ;Enabling global interrupts (maskable and non-maskable)
                andcc   #%10111111              ;One time enable XIRQ interrupt request
                                                ;See PWR_LO_ISR.asm
                cli                             ;Global enable maskable interrupts

                ;Main loop (iterated twice - one time for each motor)
MainLoop:       tst     MainLoopCounter         ;by loading the counter
                beq     MainLoop                ;Wait for interrupts events

                mAXIS_SELECT                    ;Initialize StructPtr and more

                ;GETDIR, Load the parameters and jump to the subroutine
                leas    -2,SP                   ;Stack space reserved for return values
                jsr     GETDIR                  ;Get direction from LS7266r1 via PORTA
                ldy     lAxStructPtr,X

;               brclr   0,SP,DIR,gd_forward

;gd_reverse:    bset    AxisStat,Y,DIR
;               bra     gd_end

;gd_forward:    bclr    AxisStat,Y,DIR

;gd_end:

                pula                            ;Restores stack pointer
                staa    AxisStat,Y              ;Saves the axis direction
                pula
                staa    EncStat,Y               ;Saves the encoder axis status flags

                ;GETSPEED, Load the parameters and jump to the subroutine
                leas    -2,SP
                ldy     lAxStructPtr,X
                pshy
                ldaa    AxisStat,Y
                ldab    AtdCh,Y
                pshd
                jsr     GETSPEED                ;Get speed from LN2907 via the ATD

                ;Store the returned values
                puld
                puly
                puld
                std     Speed,Y

                ;PID, Load the parameters and jump to the subroutine
                leas    -8,SP
                ldy     lAxStructPtr,X
                ldd     Speed,Y
                pshd
                ldd     SpeedRef,Y
                pshd
                ldd     Ek,Y
                pshd
                ldd     Uk+2,Y
                pshd
                ldd     Uk+0,Y
                pshd
                jsr     PID

                ;Store the returned values
                leas    10,SP                   ;Deallocate stack param space
                ldy     lAxStructPtr,X
                puld
                std     Uk+0,Y
                puld
                std     Uk+2,Y
                puld
                std     Duty,Y
                puld
                std     Ek,Y

                ;GETPOS, Load the parameters and jump to the subroutine
```

```
                IF tGETPOS==1
                jsr     GETPOS

                ;Store the returned values
                ldy     lAxStructPtr,X
                puld
                std     Position+0,Y
                puld
                std     Position+2,Y
                pula
                staa    EncStat,Y
                ENDIF

                ;GETPWRLEVEL, Load the parameters and jump to the subroutine
                leas    -1,SP                   ;Stack space reserved for return values
                ldab    BatStat                 ;Loading parameter
                pshb
                ldab    #ATDCH_B                ;Loading parameter
                pshb
                jsr     GETPWRLEVEL             ;Go to sub routine
                puld                            ;Remove parameter from the stack
                pula                            ;Remove the return value
                staa    BatStat                 ;Store updated battery status register

                dec     MainLoopCounter
                lbra    MainLoop

;;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
                NOLIST

;-------------------------------------------------------------------------------
;File name:             GETPOS.asm
;
;Description:           Return the position values of the selected motor.
;                       The value is obtained from one of the encoder IC counters.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
GETPOS:

;Stack memory allocation
                OFFSET  0
;lVar1          ds.b    1
;lvar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
rPosition       ds.b    4                       ;Return values
rEnc_Stat       ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM  Y,nBytesStorage,$00    ;Clear local storage of undefined values

;Subroutine code
                ;Reset byte pointer
                movb    #ALL,DDRA               ;Set PORTA as output
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,RLD|BIT0          ;Write RLD register, byte pointer reset
                bset    PORTB,RD|WR|CD          ;Select control mode, read and write disable
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Transferring counter to output latch
                movb    #ALL,DDRA               ;Set PORTA as output
                clr     PORTA
                bset    PORTA,RLD|BIT4
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Reset byte pointer
                movb    #ALL,DDRA               ;Set PORTA as output
                clr     PORTA                   ;Reset PORTA
                bset    PORTA,RLD|BIT0          ;Write RLD register, byte pointer reset
                bset    PORTB,RD|WR|CD          ;Select control mode, read and write disable
                bclr    PORTB,WR                ;Pulse the WR pin Low
                nop
                bset    PORTB,WR                ;Pulse the WR pin High

                ;Read output latch
                movb    #NONE,DDRA              ;Set PORTA as input
                bclr    PORTB,CD                ;Select data mode
                leay    rPosition+3,X
                ldaa    #$03
gp_loop:        bclr    PORTB,RD                ;3 bytes latched onto data bus
                nop
                ldab    PORTA                   ;Read byte from PORTA
                bset    PORTB,RD                ;Close output latch
                stab    0,Y                     ;Store LSB byte first
                dey
                dbne    A,gp_loop
                clr     0,Y                     ;Clear MSB
                bset    PORTB,CD                ;Select control mode

;               ldx     rPosition+0,X
;               ldy     rPosition+2,X

                ;Read status flag
                movb    #NONE,DDRA              ;Set PORTA as input
                bset    PORTB,RD|WR|CD          ;Select control mode
                bclr    PORTB,RD                ;Enable read on data bus
                nop
                ldab    PORTA                   ;Read encoder status flag
                bset    PORTB,RD                ;Disable read
                stab    rEnc_Stat,X             ;Push into stack

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             GETDIR.asm
;
;Description:           Return with the direction of the selected motor.
;                       Direction is obtained from the encoder IC flag register.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
GETDIR:

;Allocate stack memory
                OFFSET  0
;lVar1          ds.b    1
;lvar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack pointer
                ds.b    2                       ;Return address
rAxisStat       ds.b    1                       ;Return values
rEncStat        ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values
                clr     rAxisStat,X             ;Initialize return values
                clr     rEncStat,X              ;Initialize return values

;Subroutine code
                sei                             ;Disable interrupt while accessing I/O ports
                movb    #NONE,DDRA              ;Set PORTA as input
                bset    PORTB,RD|WR             ;Read and write disable
                bset    PORTB,CD                ;Select control mode

                bclr    PORTB,RD                ;Enable read on data bus
                movb    PORTA,rEncStat,X        ;Read encoder status flag
                bset    PORTB,RD                ;Pulse the RD pin High
                cli                             ;Reenabling interrupts

                brclr   rEncStat,X,BIT5,GD_Fwd  ;Logical test on bit-5 to determine direction

GD_Rev:         bset    rAxisStat,X,DIR
                bra     GD_End

GD_Fwd:         bclr    rAxisStat,X,DIR

;Deallocate stack memory
GD_End:         mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             GETSPEED.asm
;
;Description:           Return the speed of the selected motor.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
GETSPEED:

;Allocate stack frame memory
                OFFSET  0
;lVar1          ds.b    1
;lvar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pGSAxisStat     ds.b    1                       ;Parameter
pGSAtdCh        ds.b    1                       ;Parameter
pGSAxStructPtr  ds.b    2                       ;Parameter
rSpeed          ds.b    2                       ;Return value
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y                     ;Clear local storage of undefined values
                mSETRAM Y,nBytesStorage,$00

;Subroutine code
                movb    pGSAtdCh,X,ATDCTL5      ;Start conversion sequence
GS_spin:        brclr   ATDSTATH,SCF,GS_spin    ;Wait for conversion sequence to complete
                ldy     #$0                     ;Clear Y
                ldab    ADR0H                   ;Add the content of ADR0H with Y
                leay    B,Y
                ldab    ADR1H                   ;Add the content of ADR1H with Y
                leay    B,Y
                ldab    ADR2H                   ;Add the content of ADR2H with Y
                leay    B,Y
                ldab    ADR3H                   ;Add the content of ADR3H with Y
                leay    B,Y
                ldab    ADR4H                   ;Add the content of ADR4H with Y
                leay    B,Y
                ldab    ADR5H                   ;Add the content of ADR5H with Y
                leay    B,Y
                ldab    ADR6H                   ;Add the content of ADR6H with Y
                leay    B,Y
                ldab    ADR7H                   ;Add the content of ADR7H with Y
                leay    B,Y
                tfr     Y,D                     ;Transferring Y to D
                lsrd                            ;Calculate average of ADR0H to ADR7H
                lsrd                            ;3 right shift = divide by 8
                lsrd                            ;Result is an unsigned 8-bit number
                brclr   pGSAxisStat,X,DIR,GS_then ;If direction is forward

GS_else:        coma                            ;Else, direction is Reverse
                comb                            ;Take two's complement of the speed
                addd    #1
```

```
GS_then:        std     rSpeed,X                ;Store speed into a 16-bit signed variable
                bra     GS_end_if               ;Done

;Deallocate stack memory
GS_end_if:      mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------
;File name:             GETPWRLEVEL.asm
;
;Description:           Returns the state of the battery power level.
;                       Empty is signaled by the XIRQ interrupt.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
GETPWRLEVEL:

;Allocate stack frame memory
                OFFSET  0
lGPLBatStat     ds.b    1                       ;Local variables
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pGPLAtdCh       ds.b    1                       ;Parameter
pGPLBatStat     ds.b    1                       ;Parameter
rGPLBatStat     ds.b    1                       ;Return value
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y                     ;Clear local storage of undefined values
                mSETRAM Y,nBytesStorage,$00
                ldab    pGPLBatStat,X
                stab    lGPLBatStat,X

;Subroutine code
                movb    pGPLAtdCh,X,ATDCTL5     ;Start conversion sequence on selected ch.
GPL_Spin:       brclr   ATDSTATH,SCF,GPL_Spin   ;Wait for conversion sequence to complete
                ldy     #$0                     ;Clear Y
                ldab    ADR0H                   ;Add the content of ADR0H with Y
                leay    B,Y
                ldab    ADR1H                   ;Add the content of ADR1H with Y
                leay    B,Y
                ldab    ADR2H                   ;Add the content of ADR2H with Y
                leay    B,Y
                ldab    ADR3H                   ;Add the content of ADR3H with Y
                leay    B,Y
                ldab    ADR4H                   ;Add the content of ADR4H with Y
                leay    B,Y
                ldab    ADR5H                   ;Add the content of ADR5H with Y
                leay    B,Y
                ldab    ADR6H                   ;Add the content of ADR6H with Y
                leay    B,Y
                ldab    ADR7H                   ;Add the content of ADR7H with Y
                leay    B,Y
                tfr     Y,D                     ;Transferring Y to D
                lsrd                            ;Calculate average of ADR0H to ADR7H
                lsrd                            ;4 right shift = divide by 16
                lsrd                            ;Result is an unsigned 7-bit number
                lsrd

                tst     lGPLBatStat,X           ;Test register on BIT7 (Empty flag)
                bmi     GPL_Empty               ;If raised, battery is empty
                stab    lGPLBatStat,X           ;Else, battery is not dead. Store value.
                bra     GPL_Update_Lvl          ;Go, update the level

;Full:          cmpb    #$AA                    ;Voltage level contained in B
;               bls     Half                    ;If, lower than 2/3 branch to Half
;               bset    lGPLBatStat,X,FULL      ;Else, update battery status register
;               bra     Update_Level            ;Done

;Half:          cmpb    #$55                    ;Voltage level contained in B
;               bls     Lo                      ;If, lower than 1/3 branch to Lo
;               bset    lGPLBatStat,X,HALF      ;Else, update battery status register
;               bra     Update_Level            ;Done
;
;Lo:            bset    lGPLBatStat,X,LO        ;Update battery status register
;               bra     Update_Level            ;Done


GPL_Empty:      stab    lGPLBatStat,X
                bset    lGPLBatStat,X,EMPTY
                bra     GPL_Update_Lvl

GPL_Update_Lvl: ldab    lGPLBatStat,X           ;Load the return value
                stab    rGPLBatStat,X

;Deallocate stack memory
GPL_End:        mUNLINK nBytesStorage
                rts                             ;Return

                NOLIST
;-------------------------------------------------------------------------------
;File name:             PID.asm
;
;Description:           Computes the speed control law of the selected motor controller.
;                       The control law implemented is a Proportional-Integral (velocity form).
;                       Result is divided by 256.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
PID:

;Allocate stack frame memory
                OFFSET  0
lScndTerm       ds.b    4                       ;Local variables
lThrdTerm       ds.b    4
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pUk1            ds.b    4                       ;Parameters
```

```
pEk1            ds.b    2
pSpeedRef       ds.b    2
pSpeed          ds.b    2
rUk0            ds.b    4                       ;Return values
rDuty           ds.b    2
rEk0            ds.b    2
                mLINK   nBytesStorage

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00

;Subroutine code
                ldd     pSpeedRef,X             ;Calculate current speed error
                subd    pSpeed,X
                std     rEk0,X                  ;Store current speed error

                ;Calculate third term
                ldy     K
                emuls
                sty     lThrdTerm+0,X
                std     lThrdTerm+2,X

                ;Calculate second term
                ldd     rEk0,X
                subd    pEk1,X
                ldy     Kc
                emuls
                sty     lScndTerm+0,X
                std     lScndTerm+2,X

                ;Add second and third terms
                leas    -4,SP                   ;Reserves space for result
                pshd                            ;Push second term
                pshy
                ldy     lThrdTerm+0,X
                ldd     lThrdTerm+2,X
                pshd                            ;Push third term
                pshy
                jsr     ADD32
                leas    8,SP
                puly
                puld

                ;Add result to previous control input
                leas    -4,SP                   ;Reserves space for result
                pshd                            ;Push
                pshy
                ldy     pUk1+0,X
                ldd     pUk1+2,X
                pshd
                pshy
                jsr     ADD32
                leas    8,SP
                puly
                puld
                sty     rUk0+0,X
                std     rUk0+2,X

                ;Test the sign of control input Uk0
                brclr     rUk0+0,X,BIT7,Positive
                brset     rUk0+0,X,BIT7,Negative

                ;Adjust duty value (positive case)
Positive:       ldd     rUk0+0,X                ;Test for saturation
                cpd     #$0000                  ;Is Uk0 [0:1] equal to $0000
                bne     MaxPos                  ;No, set it max forward
                brset   rUk0+2,X,BIT7,MaxPos    ;Is duty greater than $7FFF, yes, set it max.
                bra     SetDuty                 ;Done

MaxPos:         ldy     #$0000
                ldd     #MAX_POS
                sty     rUk0+0,X
                std     rUk0+2,X
                bra     SetDuty

                ;Adjust duty value (negative case)
Negative:       ldd     rUk0+0,X                ;Test for saturation
                cpd     #$FFFF                  ;Is Uk0 [0:1] equal to $FFFF
                bne     MaxNeg                  ;No, set it max reverse
                brclr   rUk0+2,X,BIT7,MaxNeg    ;Is duty smaller than $8000, yes, set it max.
                bra     SetDuty                 ;No, done

MaxNeg:         ldy     #$FFFF
                ldd     #MAX_NEG
                sty     rUk0+0,X
                std     rUk0+2,X
                bra     SetDuty

SetDuty:        ldd     rUk0+2,X
                std     rDuty,X                 ;Set duty value based on current control input

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------------------
;File name:             CONTROL_INPUT.asm
;
;Description:           Apply the control signal to the selected motor by adjusting its assigned
;                       PWM duty register. Only the most significant byte of the duty is used.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------------------
                LIST
CONTROL_INPUT:

;Stack memory allocation
                OFFSET  0
lTempPORTP      ds.b    1                       ;Local variables
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pPwdtyPtr       ds.b    2                       ;Parameters
pPolarity1      ds.b    1
pPolarity2      ds.b    1
pDuty           ds.b    2
```

```
pCopyPORTP      ds.b    1
;rResult2       ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation
;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values
;Subroutine code
                ldab    pCopyPORTP,X            ;Local copy of PORTP
                stab    lTempPORTP,X
                ldd     pDuty,X                 ;Load the MSB of duty
                bpl     ci_forward              ;Branch if positive

ci_reverse:     ldd     #0                      ;Else the duty is negative
                subd    pDuty,X                 ;Negate to get absolute value of duty
                ldab    pPolarity1,X            ;Set polarity of motor negative (set bit)
                orab    lTempPORTP,X
                stab    lTempPORTP,X
                ldab    pPolarity2,X
                comb
                andb    lTempPORTP,X
                bra     ci_rts

ci_forward:     ldab    pPolarity1,X            ;The duty is positive
                comb                            ;Set polarity of motor positive (clear bit)
                andb    lTempPORTP,X
                stab    lTempPORTP,X
                ldab    pPolarity2,X
                orab    lTempPORTP,X

ci_rts:         stab    PORTP
                staa    [pPwdtyPtr,X]           ;Adjust speed with new duty

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------------------
;File name:             SETSPEEDREF.asm
;
;Description:           The function adjust the desired motor speeds.
;                      The new reference speeds are received via the serial communication port.
;                       The reference speeds are cast into two 16-bits integers.
;                       The packets have the form:
;                       < S | Left motor speed ASCII value | Right motor speed ASCII value >
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------------------
                LIST
SETSPEEDREF:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
lSSChar1        ds.b    1
lSSChar2        ds.b    1
lSSChar3        ds.b    1
lSSChar4        ds.b    1
lSSChar5        ds.b    1
lSSChar6        ds.b    1
lSSChar7        ds.b    1
lSSChar8        ds.b    1
lSSEOT          ds.b    1
lSSINDEX        equ     *
lXSpeedRef      ds.b    2
lYSpeedRef      ds.b    2
nBytesStorage   set     *
;               ds.b    2                       ;Saved D accumulator
;               ds.b    2                       ;Saved Y accumulator
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
;pParam1        ds.b    1                       ;Parameters
;pParam2        ds.b    1
;rVar1          ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0       ;Clear local storage of undefined values

;Subroutine code
                ldy     #lSSINDEX               ;Empty the message queue
                clra
SS_ReadNext:    psha
                pshy
                leas    -1,SP                   ;Reserves some space on stack for return value
                jsr     GETCHAR                 ;One character at a time
                pulb
                puly
                pula
                stab    A,X
                inca
                dbne    Y,SS_ReadNext

                leas    -2,SP                   ;Get X SpeedRef
                ldd     lSSChar3,X
                pshd
                ldd     lSSChar1,X
                pshd
                jsr     ATOHW
                puld
                puld
                puld
                std     lXSpeedRef,X

                leas    -2,SP                   ;Get Y SpeedRef
                ldd     lSSChar7,X
                pshd
                ldd     lSSChar5,X
                pshd
                jsr     ATOHW
                puld
                puld
                puld
                std     lYSpeedRef,X

                ldy     #XAxisStructPtr         ;Update X axis reference speed
                ldd     lXSpeedRef,X
```

```
                std     SpeedRef,Y

                ldy     #YAxisStructPtr         ;Update X axis reference speed
                ldd     lYSpeedRef,X
                std     SpeedRef,Y

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------
;File name:             SETMOTORENABLE.asm
;
;Description:           The function enable/disable the power to the motors.
;                       The desired action is received via the serial communication port.
;                       The packet have the form < E | 0 or 1 hex value>
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
SETMOTORENABLE:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
lSMEChar1       ds.b    1
lSMEOT          ds.b    1
lSMEINDEX       equ     *
nBytesStorage   set     *
;               ds.b    2                       ;Saved D accumulator
;               ds.b    2                       ;Saved Y accumulator
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
;pParam1        ds.b    1                       ;Parameters
;pParam2        ds.b    1
;rVar1          ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0       ;Clear local storage of undefined values

;Subroutine code
                ldy     #lSMEINDEX              ;Empty the message queue
                clra
SME_ReadNext:   psha
                pshy
                leas    -1,SP                   ;Reserves some space on stack for return value
                jsr     GETCHAR                 ;One character at a time
                pulb
                puly
                pula
                stab    A,X
                inca
                dbne    Y,SME_ReadNext

                leas    -1,SP
                ldab    lSMEChar1,X             ;Identification of the argument
                cmpb    #'0'
                beq     SME_Disable
                cmpb    #'1'
                beq     SME_Enable

                bra     SME_End                 ;Message unrecognized, do nothing

SME_Enable:     bset    PWEN,PWEN2|PWEN1|PWEN0  ;Enable PWM

                movb    #$7F, PWDTY1
                movb    #$7F, PWDTY0
                movb    #%01100000,PORTP
                bra     SME_End

SME_Disable:    bclr    PWEN,PWEN2|PWEN1|PWEN0  ;Disable PWM

;Deallocate stack memory
SME_End:        mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------
;File name:             SETPIDGAIN.asm
;
;Description:           The function adjust the PID gains of the motor controllers.
;                       The desired values are received via the serial communication port.
;                       The gains are cast into 16-bit integers. The packet have the form:
;                       < G | Proportional gain ASCII value | Integral gain ASCII value >
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
SETPIDGAIN:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
lSPGChar1       ds.b    1
lSPGChar2       ds.b    1
lSPGChar3       ds.b    1
lSPGChar4       ds.b    1
lSPGChar5       ds.b    1
lSPGChar6       ds.b    1
lSPGChar7       ds.b    1
lSPGChar8       ds.b    1
lSPGEOT         ds.b    1
lSPGINDEX       equ     *
lProportional   ds.b    2
lIntegral       ds.b    2
nBytesStorage   set     *
;               ds.b    2                       ;Saved D accumulator
;               ds.b    2                       ;Saved Y accumulator
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
;pParam1        ds.b    1                       ;Parameters
;pParam2        ds.b    1
```

```
;rVar1          ds.b    1                   ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage       ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0   ;Clear local storage of undefined values

;Subroutine code
                ldy     #lSPGINDEX          ;Empty the message queue
                clra
SPG_ReadNext:   psha
                pshy
                leas    -1,SP               ;Reserves some space on stack for return value
                jsr     GETCHAR             ;One character at a time
                pulb
                puly
                pula
                stab    A,X
                inca
                dbne    Y,SPG_ReadNext

                leas    -2,SP               ;Get Proportional gain
                ldd     lSPGChar3,X
                pshd
                ldd     lSPGChar1,X
                pshd
                jsr     ATOHW
                puld
                puld
                puld
                std     lProportional,X

                leas    -2,SP               ;Get Integral gain
                ldd     lSPGChar7,X
                pshd
                ldd     lSPGChar5,X
                pshd
                jsr     ATOHW
                puld
                puld
                puld
                std     lIntegral,X

                ldy     #K                  ;Update Proportional gain
                ldd     lProportional,X
                std     0,Y

                ldy     #Kc                 ;Update Integral gain
                ldd     lIntegral,X
                std     0,Y

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             SENDPWRLEVEL.asm
;
;Description:           The function sends via the serial communication port the voltage
;                       level of the battery. The voltage level is cast to an 8-bit integer.
;                       The power level packets have the form < V | 8-bit hex value | 0 | 0 | 0
>.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
SENDPWRLEVEL:

;Allocate stack frame memory
                OFFSET  0                   ;Local variables
;lVar1          ds.b    1
;lvar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                   ;Previous stack frame pointer
                ds.b    2                   ;Return address
pBatStat        ds.b    1                   ;Parameters
                mLINK   nBytesStorage       ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0   ;Clear local storage of undefined values

;Subroutine code
                ldab    #'<'                ;Forming message packet
                pshb
                jsr     PUTCHAR
                pulb

                ldab    #'V'
                pshb
                jsr     PUTCHAR
                pulb

                ldab    pBatStat,X
                pshb
                jsr     PUTCHAR
                pulb

                ldab    #'0'                ;Padding
                pshb
                jsr     PUTCHAR
                pulb

                ldab    #'0'                ;Padding
                pshb
                jsr     PUTCHAR
                pulb

                ldab    #'0'                ;Padding
                pshb
                jsr     PUTCHAR
```

```
                pulb

                ldab    #'>'
                pshb
                jsr     PUTCHAR
                pulb

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts

                NOLIST
;-------------------------------------------------------------------------------
;File name:             SENDACTUALSPEED.asm
;
;Description:           The function sends via the serial communication port the actual speed
;                       of both motors. The speed is cast into two 16-bit integers.
;                       The speed packet have the form:
;                       < S | Left motor speed hex value | Left motor speed hex value >
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
SENDACTUALSPEED:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
;lVar1          ds.b    1
;lvar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pSASSpeedX      ds.b    2                       ;Parameters
pSASSpeedY      ds.b    2                       ;Parameters
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0       ;Clear local storage of undefined values

;Subroutine code
                ldab    #'<'                    ;Forming message packet
                pshb
                jsr     PUTCHAR
                pulb

                ldab    #'S'                    ;Loading Message identifier
                pshb
                jsr     PUTCHAR
                pulb

                tfr     X,Y
                ldaa    #4
SASLoop:        pshy
                psha
                ldab    pSASSpeedX,Y
                pshb
                jsr     PUTCHAR
                pulb
                pula
                puly
                iny
                dbne    A,SASLoop

                ldab    #'>'
                pshb
                jsr     PUTCHAR
                pulb

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts

                NOLIST


;-------------------------------------------------------------------------------
;File name:             PUTSTRG.asm
;
;Description:           The function puts a string into the transmit queue of the serial
;                       communication interface.
;                   Adapted from: Doughman, Programming the Motorola M68Hc12 Family, 2000.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
PUTSTRG:

;Allocate stack frame memory
                OFFSET  0
;lVar1          ds.b    2                       ;Local variables
;lVar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pStrgPtr        ds.b    2                       ;Parameters
;pParam2        ds.b    1
;rVar1          ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y                     ;Clear local storage of undefined values
                mSETRAM Y,nBytesStorage,$00

;Subroutine code
                ldy     pStrgPtr,X
Putstr:         ldab    1,Y+
                beq     Done
                pshy
                pshb
                jsr     PUTCHAR
```

```
                pulb
                puly
                bra     Putstr

;Deallocate stack memory
Done:           mUNLINK nBytesStorage           ;Deallocate stack frame memory
                rts

                NOLIST
;-------------------------------------------------------------------------------
;File name:             PUTCHAR.asm
;
;Description:           The function puts a char into the transmit queue of the serial
;                       communication interface. The queue pointer is incremented accordingly.
;                   Adapted from: Doughman, Programming the Motorola M68Hc12 Family, 2000.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
PUTCHAR:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
;lVar1          ds.b    1
;lVar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pPChar          ds.b    1                       ;Parameters
;pParam2        ds.b    1
;rVar1          ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y                     ;Clear local storage of undefined values
                mSETRAM Y,nBytesStorage,$00

;Subroutine code
TxChk:          tst     TxBAvail                ;Any room left in the Tx queue?
                beq     TxChk                   ;No. just wait here till it is.
                ldy     #TxBuffPtr              ;Point to the physical start of the Tx queue.
                ldaa    TxIn                    ;Get the index to the next available spot.
                ldab    pPChar,X                ;Load the character
                stab    A,Y                     ;Put the character in.
                inca                            ;Point to the next available spot.
                cmpa    #TXBUFSIZE              ;Go past the end of the queue?
                blo     NopcWrap                ;No.
                clra                            ;Yes. wrap around to the start.
NopcWrap:       staa    TxIn                    ;Update the queue index
                dec     TxBAvail                ;One less byte available in the Tx queue.
                bset    SC0CR2,#TIE             ;Enable transmitter interrupts.

;Deallocate stack memory
                mUNLINK nBytesStorage           ;Deallocation of stack frame
                rts                             ;Done.
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             GETCHAR.asm
;
;Description:           The function gets a char from the received queue of the serial
;                       communication interface. The queue pointer is decremented accordingly.
;                   Adapted from: Doughman, Programming the Motorola M68Hc12 Family, 2000.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
GETCHAR:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
;lVar1          ds.b    1
;lVar2          ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
;pParam1        ds.b    1                       ;Parameters
;pParam2        ds.b    1
rChar           ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values

;Subroutine code
RxChk:          ldab    #RXBUFSIZE              ;Any characters available?
                subb    RxBAvail
                beq     RxChk                   ;No. just wait until some are.
                ldy     #RxBuffPtr              ;Point to the physical start of the Rx queue.
                ldaa    RxOut                   ;Get the index to the next available character
in the Rx queue.
                movb    A,Y,rChar,X             ;Get the character.
                inca                            ;Point to the next location in the queue.
                cmpa    #RXBUFSIZE              ;Reached the end of the queue?
                blo     NogcWrap                ;No.
                clra                            ;Yes. wrap to the start.
NogcWrap:       staa    RxOut                   ;Update the queue index.
                inc     RxBAvail                ;We removed a character from the queue
                                                ;there's 1 more available.

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             ATOHB.asm
;
;Description:           The function converts an ASCII number to an hexadecimal byte.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
ATOHB:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
;lVar1          ds.b    1
;lVar2          ds.b    1
nBytesStorage   set     *
;               ds.b    2                       ;Saved D accumulator
;               ds.b    2                       ;saved Y index register
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pHBChar1        ds.b    1                       ;Parameters
pHBChar2        ds.b    1
rHexByte        ds.b    1                       ;Return values
;rVar2          ds.b    1
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values

;Subroutine code
                ldab    pHBChar1,X              ;Get first digit
                subb    #$30
                cmpb    #$09
                bls     ShiftNibble
                subb    #$07
ShiftNibble:    aslb
                aslb
                aslb
                aslb
                stab    rHexByte,X
                ldab    pHBChar2,X              ;Get next digit
                subb    #$30
                cmpb    #$09
                bls     MergeNibbles
                subb    #$07
MergeNibbles:   andb    #%00001111              ;Clear high nibble
                orab    rHexByte,X
                stab    rHexByte,X

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             ATOHW.asm
;
;Description:           The function converts an ASCII number to an hexadecimal word.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
ATOHW:

;Allocate stack frame memory
                OFFSET  0                       ;Local variables
lMSB            ds.b    1
lLSB            ds.b    1
nBytesStorage   set     *
;               ds.b    2                       ;Saved D accumulator
;               ds.b    2                       ;saved Y index register
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pHWChar1        ds.b    1                       ;Parameters
pHWChar2        ds.b    1
pHWChar3        ds.b    1
pHWChar4        ds.b    1
rHexWord        ds.b    2                       ;Return values
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,0       ;Clear local storage of undefined values

;Subroutine code
                des
                ldd     pHWChar3,X
                pshd
                jsr     ATOHB
                puld
                pulb                            ;Least significant byte stored in accumulator B
                stab    lLSB,X

                des
                ldd     pHWChar1,X
                pshd
                jsr     ATOHB
                puld
                pula                            ;Most significant byte is in accumulator A

                staa    lMSB,X

                ldaa    lMSB,X
                ldab    lLSB,X
                std     rHexWord,X

;Deallocate stack memory
                mUNLINK nBytesStorage
                rts
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             ADD32.asm
;
;Description:           The function adds two 32 bit integers
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
ADD32:

;Allocate stack memory
                OFFSET  0                       ;Local variables
lCCR            ds.b    1
lCounter        ds.b    1
nBytesStorage   set     *
                ds.b    2                       ;Previous stack frame pointer
                ds.b    2                       ;Return address
pVar1           ds.b    4                       ;Parameters
pVar2           ds.b    4
rResult         ds.b    4                       ;Return value
                mLINK   nBytesStorage           ;Stack frame allocation

;Initialization
                tfr     X,Y
                mSETRAM Y,nBytesStorage,$00     ;Clear local storage of undefined values

;Subroutine code
                leay    pVar1+3,X               ;Y points to second variable
                movb    #4,lCounter,X
                clc
                tpa
                staa    lCCR,X                  ;Save a copy of CCR before looping
AddLoop:        ldaa    lCCR,X                  ;Retrieve last carry flag
                tap                             ;Transfer A to CCR
                ldab    0,Y                     ;Load byte from first variable
                adcb    4,Y                     ;Add byte from second variable
                tpa                             ;Transfer CCR to A
                stab    8,Y                     ;Store partial result
                staa    lCCR,X                  ;Store Condition codes in Memory
                dey                             ;Increment pointers
                dec     lCounter,X
                bne     AddLoop                 ;Continue if not finished
                brset   lCCR,X,#$03,Underflow   ;Test C and V bit for underflow
                brset   lCCR,X,#$02,Overflow    ;Test V bit for two's complement overflow
                bra     ADD32_End               ;No overflow, no underflow return

Underflow:      ldy     #$8000                  ;Underflow caused by addition of 2 negative
numbers
                ldd     #$0001                  ;Saturate result to min value $80000001
                bra     Replace

Overflow:       ldy     #$7FFF                  ;Overflow caused by addition of 2 positive numbers
                ldd     #$FFFF                  ;Saturate result to max value $7FFFFFFF

Replace:        sty     rResult+0,X
                std     rResult+2,X

;Deallocate stack memory
ADD32_End:      mUNLINK nBytesStorage
                rts
                NOLIST
;-------------------------------------------------------------------------------
;File name:             SCI_ISR.asm
;
;Description:           Serial communication interrupt service routine.
;                       Dispatch the message received.
;                   Adapted from: Doughman, Programming the Motorola M68Hc12 Family, 2000.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
SCI_ISR:

;Allocate stack memory
                mLINK   $00                     ;Stack frame allocation

;Subroutine code
                ;Check interrupt source
                brclr   SC0CR2,#RIE,ChkRxInts   ;Rx interrupts enabled?
                brset   SC0SR1,#RDRF,RxIRQ      ;yes. if RDRF flag set, service Rx interrupt.
ChkRxInts:      brclr   SC0CR2,#TIE,NoSCIInt    ;Tx interrupts enabled?
                brset   SC0SR1,#TDRE,TxIRQ      ;Yes. if TDRE is set, service Tx interrupt
NoSCIInt:       bra     SciRTI                  ;Should never get here, return w/o any action.

                ;Transmetter interrupt service routine
TxIRQ:          ldy     #TxBuffPtr              ;Point to the physical start of the Tx queue.
                ldaa    TxOut                   ;Get the index for the next character to send.
                ldab    A,Y                     ;Get the data.
                stab    SC0DRL                  ;Send it.
                inca                            ;Advance to next character to send.
                cmpa    #TXBUFSIZE              ;Reached the end of the queue?
                blo     NoTxWrap                ;No.
                clra                            ;Yes. wrap to the start.
NoTxWrap:       staa    TxOut                   ;Update the queue index.
                inc     TxBAvail                ;One more byte available in the queue.
                cmpa    TxIn                    ;TxIn = TxOut?
                bne     SciRTI                  ;No. more characters to send.
                bclr    SC0CR2,#TIE             ;Yes. queue is empty turn off TDRE interrupts.
                bra     SciRTI

                ;Receiver interrupt service routine
RxIRQ:          tst     RxBAvail                ;Any room left in the Rx queue?
                beq     Buffull                 ;No. just throw the character away.
                dec     RxBAvail                ;Yes, there'll be one less now.
                ldy     #RxBuffPtr              ;Point to the physical start of the Rx queue.
                ldaa    RxIn                    ;Get the index for the next available queue loc.
                ldab    SC0DRL                  ;Get the received character.
                stab    A,Y                     ;Place it in the queue.
                inca                            ;Next available queue location.
                cmpa    #RXBUFSIZE              ;Wrap around to start of queue?
                blo     NoRxWrap                ;No. just update the index.
                clra                            ;Yes, start at beginning of queue.
NoRxWrap:       staa    RxIn                    ;Update the next available queue location index.
                cmpb    #'>'                    ;Has it reached the end of the msg transmission.
```

```
                beq     RxDispatch              ;Yes, dispatch message received
                bra     SciRTI                  ;No, return w/o any action

Buffull:        ldab    SC0DRL                  ;Queue full, get character, throw it away.
                bra     SciRTI                  ;Done

                ;Dispatch received message
RxDispatch:     leas    -1,SP                   ;Reserves some space on stack for return char
                jsr     GETCHAR                 ;Get message identifier (SOH)
                pulb                            ;Get rid of it
                leas    -1,SP                   ;Reserves some space on stack for return char
                jsr     GETCHAR                 ;Loading msg identifier in accumulator B.
                pulb
                cmpb    MsgSpeedRef             ;Compare message identifier
                beq     SpdRef                  ;Yes, update speed set points
                cmpb    MsgPIDGain              ;Compare message identifier
                beq     PIDGain
                cmpb    MsgMotorEnable          ;Compare message identifier
                beq     MotorEnable

                ldaa    RxIn                    ;Message unrecognized, queue may be corrupted
                staa    RxOut                   ;Queue pointer for restoring data integrity

                movb    #RXBUFSIZE,RxBAvail     ;Reset buffer
                bra     SciRTI

                ;Message jump table
SpdRef:         jsr     SETSPEEDREF
                bra     SciRTI
PIDGain:        jsr     SETPIDGAIN
                bra     SciRTI
MotorEnable:    jsr     SETMOTORENABLE


;Deallocate stack memory
SciRTI:         mUNLINK $00
                rti

                NOLIST
;-------------------------------------------------------------------------------
;File name:             ISR_HANDLER.asm
;
;Description:           Default interrupt handler for unimplemented service routine.
;                       Recovers gracefully from an unimplemented interrupt by redirecting
;                       to the reset vector.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
ISR_HANDLER:

;Allocate stack memory
                mLINK   $00
                ldd     #HANDLERStrg
                pshd
                jsr     PUTSTRG
                puld

;Deallocate stack memory
                mUNLINK  $00
                rti

                NOLIST
;-------------------------------------------------------------------------------
;File name:             PWR_LO_ISR.asm
;
;Description:           External interrupt service routine.
;                       Interrupt is latched when battery voltage level is low.
;                       A warning message is sent via the serial communication interface.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
PWR_LO_ISR:

;Allocate stack frame memory
                mLINK   $00                     ;Stack frame allocation

;Subroutine code
                bset    2,SP,BIT6               ;Mask any further XIRQ - CAUTION!
                                                ;Stacked CCR register modified
                ;Print on terminal a warning
                brset   BatStat,BIT7,PLI_End    ;If flag already raised do nothing
                bset    BatStat,BIT7            ;Else, raise the flag and,
                ldd     #PWREMPTYStrg           ;Display a warning to the terminal
                pshd
                jsr     PUTSTRG
                puld

;Deallocate stack memory
PLI_End:        mUNLINK  $00
                rti                             ;Return and restore CPU state

                NOLIST
;-------------------------------------------------------------------------------
;File name:             RTI_ISR.asm
;
;Description:           Real-time interrupt service routine. Motor control signals are updated.
;                       Power level and actual speeds are transmitted via the serial
;                       communication interface.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
RTI_ISR:

;Allocate stack frame memory
                mLINK   $00                     ;Stack frame allocation

;Subroutine code
                bset    RTIFLG,RTIF             ;Acknowledge the interrupt, resume interrupt
```

```
                ;Reset the main loop counter
                movb    #2,MainLoopCounter      ;Upon return of the ISR, allows to calculate
                                                ;the next control signals (X and Y channel)
                ;Update of channel X (left motor)
rti_updateX:    ldy     #XAxisStructPtr
                ldaa    PORTP                   ;Load state of PORT P
                psha
                ldd     Duty,Y                  ;Load signed duty (Uk)
                pshd
                ldaa    Polarity2,Y             ;Load PORT P bit mask for ST L298 IN2 pin
                psha
                ldaa    Polarity1,Y             ;Load PORT P bit mask for ST L298 IN2 pin
                psha
                ldd     PwdtyPtr,Y              ;Load channel X duty register
                pshd
                ;jsr    CONTROL_INPUT           ;Calculate control signal sign and magnitude
                leas    7,SP

                ;Update of channel Y (right motor)
rti_updateY:    idy     #YAxisStructPtr
                ldaa    PORTP                   ;Load state of PORT P
                psha
                ldd     Duty,Y                  ;Load signed duty (Uk)
                pshd
                ldaa    Polarity2,Y             ;Load PORT P bit mask for ST L298 IN2 pin
                psha
                ldaa    Polarity1,Y             ;Load PORT P bit mask for ST L298 IN1 pin
                psha
                ldd     PwdtyPtr,Y              ;Load channel Y duty register
                pshd
                ;jsr    CONTROL_INPUT           ;Calculate control signal sign and magnitude
                leas    7,SP

                ;Send power level
                ldab    BatStat                 ;Loading parameter
                pshb
                ;jsr     SENDPWRLEVEL
                leas    1,SP                    ;Clean stack

                ;Send actual speed
                ldy     #YAxisStructPtr
                ldd     Speed,Y                 ;Loading parameter
                pshd
                ldy     #XAxisStructPtr
                ldd     Speed,Y                 ;Loading parameter
                pshd
                jsr     SENDACTUALSPEED
                leas    4,SP                    ;Clean stack

;Deallocate stack frame
end_if:         mUNLINK $00
                rti
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             INT_VEC_JMP_TBL.inc
;
;Description:           Definition of the interrupt vector table and service routine handlers.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
iReserved0      ds.w    1                       ;$F7C0:$F7C1 - Reserved not implemented
iReserved1      ds.w    1                       ;$F7C2:$F7C3 - Reserved not implemented
iReserved2      ds.w    1                       ;$F7C4:$F7C5 - Reserved not implemented
iReserved3      ds.w    1                       ;$F7C6:$F7C7 - Reserved not implemented
iReserved4      ds.w    1                       ;$F7C8:$F7C9 - Reserved not implemented
iReserved5      ds.w    1                       ;$F7CA:$F7CB - Reserved not implemented
iReserved6      ds.w    1                       ;$F7CC:$F7CD - Reserved not implemented
iReserved7      dc.w    ISR_HANDLER             ;$F7CE:$F7CF - Not implemented
iUserBDLC       dc.w    ISR_HANDLER             ;$F7D0:$F7D1 - $FFD0:$FFD1 BDLC (J1850)
iUserAtoD       dc.w    ISR_HANDLER             ;$F7D2:$F7D3 - $FFD2:$FFD3 ATD
iReserved10     ds.w    1                       ;$F7D4:$F7D5 - $FFD6:$FFD7 Reserved
iUserSCI0       dc.w    SCI_ISR                 ;$F7D6:$F7D7 - $FFD6:$FFD7 SCI Channel 0

iUserSPI0       dc.w    ISR_HANDLER             ;$F7D8:$F7D9 - $FFD8:$FFD9 SPI
iUserPAccEdge   dc.w    ISR_HANDLER             ;$F7DA:$F7DB - $FFDA:$FFDB Pulse Acc. Input Edge
iUserPAccOvf    dc.w    ISR_HANDLER             ;$F7DC:$F7DD - $FFDC:$FFDD Pulse Acc. Overflow
iUserTimerOvf   dc.w    ISR_HANDLER             ;$F7DE:$F7DF - $FFDE:$FFDF Timer Overflow
iUserTimerCh7   dc.w    ISR_HANDLER             ;$F7E0:$F7E1 - $FFE0:$FFE1 Timer Channel 7
iUserTimerCh6   dc.w    ISR_HANDLER             ;$F7E2:$F7E3 - $FFE2:$FFE3 Timer Channel 6
iUserTimerCh5   dc.w    ISR_HANDLER             ;$F7E4:$F7E5 - $FFE4:$FFE5 Timer Channel 5
iUserTimerCh4   dc.w    ISR_HANDLER             ;$F7E6:$F7E7 - $FFE6:$FFE7 Timer Channel 4
iUserTimerCh3   dc.w    ISR_HANDLER             ;$F7E8:$F7E9 - $FFE8:$FFE9 Timer Channel 3
iUserTimerCh2   dc.w    ISR_HANDLER             ;$F7EA:$F7EB - $FFEA:$FFEB Timer Channel 2
iUserTimerCh1   dc.w    ISR_HANDLER             ;$F7EC:$F7ED - $FFEC:$FFED Timer Channel 1
iUserTimerCh0   dc.w    ISR_HANDLER             ;$F7EE:$F7EF - $FFEE:$FFEF Timer Channel 0
iUserRTI        dc.w    RTI_ISR                 ;$F7F0:$F7F1 - $FFF0:$FFF1 Real Time Interrupt
iUserIRQ        dc.w    ISR_HANDLER             ;$F7F2:$F7F3 - $FFF2:$FFF3 IRQ
iUserXIRQ       dc.w    PWR_LO_ISR              ;$F7F4:$F7F5 - $FFF4:$FFF5 XIRQ
iUserSWI        dc.w    ISR_HANDLER             ;$F7F6:$F7F7 - $FFF6:$FFF7 SWI
iUserTRAP       dc.w    ISR_HANDLER             ;$F7F8:$F7F9 - $FFF8:$FFF9 Illegal Opcode Trap
iUserCOPFail    dc.w    ISR_HANDLER             ;$F7FA:$F7FB - $FFFA:$FFFB COP Failure Reset
iUserCOPMonFail dc.w    ISR_HANDLER             ;$F7FC:$F7FD - $FFFC:$FFFD Clock Mon. Fail Reset
iUserReset      dc.w    Entry                   ;$F7FE:$F7FF - $FFFE:$FFFF Reset
```

```
                NOLIST
;-------------------------------------------------------------------------------
;File name:             macros.inc
;
;Description:           Macro definition file.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2005
;
;Processor:             Motorola MC68HC912B32
;-------------------------------------------------------------------------------
                LIST
```

```
;Prototype:     mLINK   nBytesStorage
mLINK:          MACRO
                pshx
                leas    -\1,SP
                tsx
                cps     #STACK_LO               ;Check lower stack boundary
;               blo     ErrorMsg
                bra      \@END_
;ErrorMsg:      swi
;               bra     \@END_
\@END_:
                ENDM

;Prototype:     mUNLINK nBytesStorage
mUNLINK:        MACRO
                leas    \1,X
                pulx
;               cps     #STACK_HI
;               bhi     ErrorMsg
;               bra     \@END_
;ErrorMsg:      swi
;               bra     \@END_
\@END_:
                ENDM

;Prototype:     mAXIS_SELECT (no arguments)
mAXIS_SELECT:   MACRO
                brclr   lAxis,X,YX,\@SelectY
                brset   lAxis,X,YX,\@SelectX
                bra     \@End_

\@SelectX:      ldy     #XAxisStructPtr
                sty     lAxStructPtr,X
                bclr    lAxis,X,BIT7
                bclr    PORTB,YX                ;Set PortB (control port for the LS7266R1)
                bra     \@End_

\@SelectY:      ldy     #YAxisStructPtr
                sty     lAxStructPtr,X
                bset    lAxis,X,BIT7
                bset    PORTB,YX                ;Set PortB (control port for the LS7266R1)
                bra     \@End_
\@End_:
                ENDM

;Prototype:     mSET_VECTOR ISR Parameter,ISR Address
mSET_VECTOR:    MACRO
                ldd     \2                      ;Load address of ISR
                pshd                            ;Push address onto stack
                ldd     \1                      ;ISR parameter
                jsr     [SetUserVector,PCR]     ;Jump at address of the callable routine
                pulx                            ;Clean up stack
                ENDM

;Prototype:     mDELAY  Amount of Delay
mDELAY:         MACRO
                ldaa    #\1
\@Wait:         deca
                bne     \@Wait
                ENDM
```

```
;Prototype:     mCLRAM  RAM_START->D, RAM_SIZE
mCLRAM:         MACRO
                tfr     \1,Y
\@Clear:        clr     1,Y+                    ;Clear one RAM location and inc pointer
                cpy     \2,X
                bne     \@Clear                 ;Loop till RAM memory is cleared
                ENDM

;Prototype:     mSETRAM address LOW boundary -> Y, size, set to value
mSETRAM:        MACRO
                ldd     #\2                     ;Load number of RAM addresses to init
                cpd     #$0                     ;Is Size = 0
                bls     \@NoneToSet             ;Yes, escape from macro
\@SetMore:      movb    #\3,1,Y+                ;No. clear one RAM location and inc idx ptr
                dbne    D,\@SetMore             ;Loop till RAM memory is cleared
\@NoneToSet:
                ENDM
```

C.2 Human-Machine Interface

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by AUMERGUI.rc
//

#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_AUMERGUI_DIALOG             102
#define ID_INDICATOR_STATUS_PANE2       102
#define ID_INDICATOR_STATUS_PANE3       103
#define ID_INDICATOR_STATUS_PANE1       104
#define IDR_MAINFRAME                   128
#define IDR_MENU1                       129
#define IDC_SLIDER_SPEED                1000
#define IDC_SLIDER_DIR                  1001
#define IDC_EDIT_RADIUS                 1004
#define IDC_EDIT_SPEED                  1005
#define IDC_STATIC_VIDEO                1006
#define IDC_MSCOMM1                     1007
#define IDC_SBARCTRL2                   1009
#define IDC_SBARCTRL3                   1010
#define IDC_POSITIONX                   1014
#define IDC_POSITIONY                   1015
#define IDC_YAW                         1019
#define IDC_PROPORTIONAL                1020
#define IDC_INTEGRAL                    1021
#define IDC_PROGRESS1                   1022
#define IDC_VOLTAGELEVEL                1022
#define IDC_BUTTONSTART                 1026
#define IDC_RADIOENABLE                 1034
#define IDC_RADIODISABLE                1035
#define IDC_BUTTONSTOP                  1037
#define IDC_SPINPOSITIONX               1038
#define IDC_SPININTEGRAL                1039
#define IDC_SPINPROPORTIONAL            1040
#define IDC_SPINPOSITIONY               1041
#define IDC_SPINTENSION                 1042
#define IDC_TENSION                     1043
#define IDC_POSITIONXACTUAL             1044
#define IDC_POSITIONYACTUAL             1045
#define IDC_EDIT1                       1046
#define IDC_EDIT2                       1047
#define IDC_EDIT_SPEEDACTUAL            1047
#define IDC_SPIN1                       1048
#define IDC_SPIN2                       1049
#define IDC_STATIC_PICTURE              1050
#define IDC_SLIDER1                     1051
#define ID_FILE_EXIT                    32771
#define ID_Menu                         32772
#define ID_CAMERA_DISPLAYPROPERTIES     32773
#define ID_CAMERA_DISPLAY               32784
#define ID_CAMERA_COMPRESSION           32785
#define ID_CAMERA_FORMAT                32786
#define ID_CAMERA_SOURCE                32787

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        134
#define _APS_NEXT_COMMAND_VALUE         32788
#define _APS_NEXT_CONTROL_VALUE         1052
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
//-----------------------------------------------------------------------------
// AUMERGUI.h : main header file for the PROJECT_NAME application
//-----------------------------------------------------------------------------

#pragma once

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols
//#include "mclmcr.h"

// CAUMERGUIApp:
// See AUMERGUI.cpp for the implementation of this class
//

class CAUMERGUIApp : public CWinApp
{
public:
    CAUMERGUIApp();

// Overrides
    public:
    virtual BOOL InitInstance();

// Implementation

    DECLARE_MESSAGE_MAP()
};

extern CAUMERGUIApp theApp;
```

```
//-----------------------------------------------------------------------------
// AUMERGUIDlg.h : header file
//-----------------------------------------------------------------------------

#pragma once
#include "afxcmn.h"
```

```
#include "afxwin.h"
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "cvcam.h"
#include "mscomm1.h"
#include "CNavigation.h"
#include "mclmcr.h"
#include "libmatrixp.h"

// CAUMERGUIDlg dialog
class CAUMERGUIDlg : public CDialog
{
// Construction
public:
	CAUMERGUIDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	enum { IDD = IDD_AUMERGUI_DIALOG };
	CSliderCtrl m_SliderSpeedCtrl;
	CSliderCtrl m_SliderDirCtrl;
	CStatic m_StaticVideoCtl;
	CImage PreviewVideo;
	CImage PreviewPicture;
	CNavigation nav;
	CMscomm1 m_mscomm;
	CString	m_input;
	CStatusBar m_bar;
	CButton m_StartButtonCtl;
	CButton m_StopButtonCtl;
	CSpinButtonCtrl m_SpinTensionCtl;
	CSpinButtonCtrl m_SpinPositionYCtl;
	CString m_RadiusValue;
	CString m_SpeedValue;
	CSpinButtonCtrl m_SpinPositionXCtl;
	CSpinButtonCtrl m_SpinProportionalCtl;
	CSpinButtonCtrl m_SpinIntegralCtl;
	CString m_PositionXValue;
	CString m_PositionYValue;
	CString m_PositionXActualValue;
	CString m_PositionYActualValue;
	CString m_TensionValue;
	CString m_ProportionalValue;
	CString m_IntegralValue;
	CString m_YawValue;
	CProgressCtrl m_VoltageLevelCtl;
	CStatic m_StaticPictureCtl;
	float m_Metrics[8];

	bool MatlabDllInit;
	void InitStaticVideo(void);
	void RefreshStaticVideo(void);
	void OnCommMscomm1();

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	virtual BOOL PreTranslateMessage(MSG* pMsg);

// Implementation
protected:
	HICON m_hIcon;
	IplImage* frameRGB;
	IplImage* pictureRGB;
	IplImage* ActualVideo;
	IplImage* ActualPicture;
	CvCapture* capture;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnPaint();
	afx_msg void OnFileExit(void);
	afx_msg void OnCameraSource();
	afx_msg void OnCameraDisplay();
	afx_msg void OnCameraCompression();
	afx_msg void OnCameraFormat();
	afx_msg void OnMouseMove(UINT nFlags, CPoint point);
	afx_msg void OnBnClickedRadiodisable();
	afx_msg void OnBnClickedRadioenable();
	afx_msg void OnEnChangeEdit1();
	afx_msg void OnBnClickedStart();
	afx_msg void OnBnClickedButtonstop();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
	DECLARE_EVENTSINK_MAP()
public:
	afx_msg void OnDeltaposSpinproportional(NMHDR *pNMHDR, LRESULT *pResult);

	afx_msg void OnDeltaposSpinintegral(NMHDR *pNMHDR, LRESULT *pResult);
	CString m_SpeedActualValue;
protected:
	virtual void OnCancel();
};



//-----------------------------------------------------------------------------
// CNavigation.h : header file
//-----------------------------------------------------------------------------

#pragma once

#include <math.h>
#include <iostream>
#include <fstream>
#include "StdAfx.h"
#include "cv.h"
#include "highgui.h"
#include "cvcam.h"
#include "libmatrixp.h"
#include "PreciseTimer.h"
using namespace std;



#define aMAX   255;
enum rs232msg {SPEED, MOTORENABLE, PIDGAIN};
enum navmode {MANUAL,AUTO};
class CNavigation
{
public:
```

```
    CNavigation(void);
    ~CNavigation(void);
    void SetWheelSpeedRef(int SpeedSetPoint, int CurvatureRadiusSetPoint, int RadiusRangeMAX);
    float GetRadiusRc(void);
    float GetSpeedVc(void);
    COleVariant SendMessage232(rs232msg Message, int param1 = 0, int param2 = 0);
    void itob(short n, char s[]);
    void SpeedtoByte(void);
    void GaintoByte(void);
    bool SetNavigationMode(navmode Mode);
    int GetNavigationMode(void);
    bool StartNavigation(int, int, int, IplImage* );

    void SetStartNavigationFcnFlag(bool );
    bool GetStartNavigationFcnFlag();
    void GetNavigationMetrics(float* );
    void SaveNavigationMetrics();
    void SetSpeedVcActual(short ,short );
    void SetPositionActual(unsigned int ,unsigned int );
    float GetSpeedVcActual();
    int GetRadiusRcSliderPos(void);
    void SaveSpeedActual(void);
    void SavePositionActual(void);
    CString GetElapsedTime();

public:
    char wlByte[2*sizeof(short)+1];
    char wrByte[2*sizeof(short)+1];
    char IntegralByte[2*sizeof(short)+1];
    char ProportionalByte[2*sizeof(short)+1];

    IplImage* frameGRAY;
    IplImage* frameRGB;
    mxUint8* ImgBuffer;
    float* OutputBuffer;
    float Metrics[10];
    int nElements;
    float out;


protected:
    float Rwheel;            //Radius of the wheels [inches]
    float L;             //Distance accross the wheels [inches]
    short wl;
    short wr;
    float Vc;
    short wlActual;
    short wrActual;
    float VcActual;
    unsigned int positionlActual;
    unsigned int positionrActual;
    float Rc;
   short Integral;
    short Proportional;
    int RcspSliderPos;
    short* pwl;
    short* pwr;
    float* pVc;
    float* pRc;
    bool steering;
    bool direction;
    int NavigationMode;
    bool StartNavigationFcnFlag;
    CPreciseTimer TickCount;
    CString    ElapsedTime;
    ofstream datafile;
};

//-----------------------------------------------------------------------------
// AUMERGUI.cpp : Defines the class behaviors for the application.
//-----------------------------------------------------------------------------

#include "stdafx.h"
#include "AUMERGUI.h"
#include "AUMERGUIDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CAUMERGUIApp

BEGIN_MESSAGE_MAP(CAUMERGUIApp, CWinApp)
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()


// CAUMERGUIApp construction

CAUMERGUIApp::CAUMERGUIApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}


// The one and only CAUMERGUIApp object

CAUMERGUIApp theApp;


// CAUMERGUIApp initialization

BOOL CAUMERGUIApp::InitInstance()
{
    // InitCommonControls() is required on Windows XP if an application
    // manifest specifies use of ComCtl32.dll version 6 or later to enable
    // visual styles.  Otherwise, any window creation will fail.
    if( !mclInitializeApplication(NULL,0))
    {
        fprintf(stderr, "Could not initialize Matlab Runtime Component.\n");
        exit(1);
    }

    InitCommonControls();

    CWinApp::InitInstance();

    AfxEnableControlContainer();
```

```
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need
    // Change the registry key under which our settings are stored
    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    CAUMERGUIDlg dlg;
    m_pMainWnd = &dlg;
    INT_PTR nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
        libmatrixpTerminate();
        mclTerminateApplication();
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}


//-----------------------------------------------------------------------------
// AUMERGUIDlg.cpp : implementation file
//-----------------------------------------------------------------------------

#include "stdafx.h"
#include "AUMERGUI.h"
#include "AUMERGUIDlg.h"
#include ".\aumerguidlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


static UINT BASED_CODE indicators[] =
{
    ID_INDICATOR_STATUS_PANE1,
    ID_INDICATOR_STATUS_PANE2
};

// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnPaint();
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    ON_WM_PAINT()
END_MESSAGE_MAP()


// CAUMERGUIDlg dialog

CAUMERGUIDlg::CAUMERGUIDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAUMERGUIDlg::IDD, pParent)
    , m_RadiusValue(_T(""))
    , m_SpeedValue(_T(""))
    , m_PositionXValue(_T("24"))
    , m_PositionYValue(_T("0"))
    , m_PositionXActualValue(_T("0.0"))
    , m_PositionYActualValue(_T("0.0"))
    , m_TensionValue(_T("60"))
    , m_YawValue(_T("0.00"))
    , m_ProportionalValue(_T("2"))
    , m_IntegralValue(_T("1"))
    , MatlabDllInit(false)
    , m_SpeedActualValue(_T("0.0"))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

    for(int i = 0; i < 8; i++)
        m_Metrics[i] = 0;

}
void CAUMERGUIDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_SLIDER_SPEED, m_SliderSpeedCtrl);
    DDX_Control(pDX, IDC_SLIDER_DIR, m_SliderDirCtrl);
    DDX_Control(pDX, IDC_STATIC_VIDEO, m_StaticVideoCtl);
    DDX_Control(pDX, IDC_MSCOMM1, m_mscomm);
    DDX_Text(pDX, IDC_EDIT_RADIUS, m_RadiusValue);
    DDX_Text(pDX, IDC_EDIT_SPEED, m_SpeedValue);
    DDX_Control(pDX, IDC_BUTTONSTART, m_StartButtonCtl);
    DDX_Control(pDX, IDC_BUTTONSTOP, m_StopButtonCtl);
    DDX_Text(pDX, IDC_POSITIONX, m_PositionXValue);
    DDX_Text(pDX, IDC_POSITIONY, m_PositionYValue);
    DDX_Control(pDX, IDC_SPINPOSITIONX, m_SpinPositionXCtl);
```

```
    DDX_Control(pDX, IDC_SPINPOSITIONY, m_SpinPositionYCtl);
    DDX_Text(pDX, IDC_POSITIONXACTUAL, m_PositionXActualValue);
    DDX_Text(pDX, IDC_POSITIONYACTUAL, m_PositionYActualValue);
    DDX_Text(pDX, IDC_YAW, m_YawValue);
    DDX_Control(pDX, IDC_SPINTENSION, m_SpinTensionCtl);
    DDX_Text(pDX, IDC_TENSION, m_TensionValue);
    DDX_Control(pDX, IDC_SPINPROPORTIONAL, m_SpinProportionalCtl);
    DDX_Control(pDX, IDC_SPININTEGRAL, m_SpinIntegralCtl);
    DDX_Text(pDX, IDC_PROPORTIONAL, m_ProportionalValue);
    DDX_Text(pDX, IDC_INTEGRAL, m_IntegralValue);
    DDX_Control(pDX, IDC_VOLTAGELEVEL, m_VoltageLevelCtl);
    DDX_Control(pDX, IDC_STATIC_PICTURE, m_StaticPictureCtl);
    DDX_Text(pDX, IDC_EDIT_SPEEDACTUAL, m_SpeedActualValue);
}

BEGIN_MESSAGE_MAP(CAUMERGUIDlg, CDialog)
    // CAUMERGUIDlg message handlers
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
    ON_WM_TIMER()
    ON_COMMAND(ID_FILE_EXIT, OnFileExit)
    ON_COMMAND(ID_CAMERA_SOURCE, OnCameraSource)
    ON_COMMAND(ID_CAMERA_DISPLAY, OnCameraDisplay)
    ON_COMMAND(ID_CAMERA_COMPRESSION, OnCameraCompression)
    ON_COMMAND(ID_CAMERA_FORMAT, OnCameraFormat)
    ON_WM_MOUSEMOVE()

    // User defined for custom user message
    ON_BN_CLICKED(IDC_BUTTONSTART, OnBnClickedStart)
    ON_BN_CLICKED(IDC_BUTTONSTOP, OnBnClickedButtonstop)
    ON_BN_CLICKED(IDC_RADIODISABLE, OnBnClickedRadiodisable)
    ON_BN_CLICKED(IDC_RADIOENABLE, OnBnClickedRadioenable)
    ON_NOTIFY(UDN_DELTAPOS, IDC_SPINPROPORTIONAL, OnDeltaposSpinproportional)

    ON_NOTIFY(UDN_DELTAPOS, IDC_SPININTEGRAL, OnDeltaposSpinintegral)
    END_MESSAGE_MAP()

BEGIN_EVENTSINK_MAP(CAUMERGUIDlg, CDialog)
    ON_EVENT(CAUMERGUIDlg, IDC_MSCOMM1, 1, OnCommMscomm1, VTS_NONE)
    END_EVENTSINK_MAP()

// CAUMERGUIDlg message handlers

void CAUMERGUIDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}


void CAUMERGUIDlg::OnPaint()
{
    // If you add a minimize button to your dialog, you will need the code below
    //  to draw the icon.  For MFC applications using the document/view model,
    //  this is automatically done for you by the framework.
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {

        CDialog::OnPaint();
        // Refresh video screen
        CDC *dc1 = m_StaticVideoCtl.GetDC();
        PreviewVideo.Show (dc1->m_hDC, 0, 0, PreviewVideo.Width(), PreviewVideo.Height(), 0,
0);
      m_StaticVideoCtl.ReleaseDC( dc1 );

        //Refresh path screen
        CDC *dc2 = m_StaticPictureCtl.GetDC();
        PreviewPicture.Show (dc2->m_hDC,      0 ,
0,PreviewPicture.Width(),PreviewPicture.Height(),0,0);
        m_StaticPictureCtl.ReleaseDC( dc2 );
    }
}

HCURSOR CAUMERGUIDlg::OnQueryDragIcon()
{
    // The system calls this function to obtain the cursor to display while the user drags
    // the minimized window.
    return static_cast<HCURSOR>(m_hIcon);
}


BOOL CAUMERGUIDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_bar.Create(this); //We create the status bar
    m_bar.SetIndicators(indicators,3); //Set the number of panes
    CRect rect;
    GetClientRect(&rect);
    //Size the two panes
    m_bar.SetPaneInfo(0,ID_INDICATOR_STATUS_PANE1, SBPS_NORMAL,rect.Width()-100);
    m_bar.SetPaneInfo(1,ID_INDICATOR_STATUS_PANE2, SBPS_NORMAL,70);
    m_bar.SetPaneInfo(2,ID_INDICATOR_STATUS_PANE3, SBPS_STRETCH,0);

    //This is where we actually draw the status bar on the screen
    RepositionBars(AFX_IDW_CONTROLBAR_FIRST,AFX_IDW_CONTROLBAR_LAST,
                                                    ID_INDICATOR_STATUS_PANE3);
```

```
// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
     CString strAboutMenu;
     strAboutMenu.LoadString(IDS_ABOUTBOX);
     if (!strAboutMenu.IsEmpty())
     {
          pSysMenu->AppendMenu(MF_SEPARATOR);
          pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
     }

}

// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);              // Set big icon
SetIcon(m_hIcon, FALSE);        // Set small icon

// Enabling Property page on start
//cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH_HEIGHT, 320240 );
//cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_SOURCE, 0 );
//cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_FORMAT, 0 );

//Initializing serial port
m_mscomm.put_OutBufferCount(0);                    //Clear the output buffer
m_mscomm.put_InBufferCount(0);                     //Clear the input buffer
m_mscomm.put_PortOpen(true);                       //open port

//Initializing the Speed slider
m_SliderSpeedCtrl.SetRangeMax(896);
m_SliderSpeedCtrl.SetRangeMin(-896);
m_SliderSpeedCtrl.SetPos(0);
m_SliderSpeedCtrl.SetTicFreq(112);
m_SpeedValue.Format("%.01f", nav.GetSpeedVc());

//Initializing the radius slider
m_SliderDirCtrl.SetRangeMax(1024);
m_SliderDirCtrl.SetRangeMin(-1024);
m_SliderDirCtrl.SetPos(0);
m_SliderDirCtrl.SetTicFreq(128);
m_RadiusValue.Format("%.1lf", nav.GetRadiusRc());

//Initializing the spin controls
//UDACCEL AccellValue;
//AccellValue.nSec = 1000;
//AccellValue.nInc = 5;
//m_SpinPositionXCtl.SetAccel(1, &AccellValue);
m_SpinPositionXCtl.SetRange(0, 48);
m_SpinPositionYCtl.SetRange(-256, 256);
m_SpinTensionCtl.SetRange(10, 200);
m_SpinProportionalCtl.SetRange(0, 100);
m_SpinIntegralCtl.SetRange(0, 100);

//Initialization of voltage level progress bar
m_VoltageLevelCtl.SetRange(0,255);
     m_VoltageLevelCtl.SetPos(255);

     //Initializing the video display
     InitStaticVideo();

     SetTimer(1,400,NULL);
     //Initializing the clock in the statusbar
     SetTimer(100,1000,NULL);   // 1000*60 msec interval
     UpdateData(false);
     return TRUE;  // return TRUE  unless you set the focus to a control
// same as double-clicking on main window close box
}

void CAUMERGUIDlg::InitStaticVideo()
{
     //SendMessage(WM_FRAME_REFRESH);
     // Initialization
     ActualVideo  = 0;
    ActualPicture = 0;
     capture = 0;

     // Enabling capture from camera
     //capture = cvCaptureFromCAM(-1);

     //Enabling capture from file
     capture = cvCaptureFromFile("VideoCeiling.avi");
     if( !capture )
          MessageBox("Could not initialize AVI capturing.");
     frameRGB = cvQueryFrame(capture);
     PreviewVideo.Create(frameRGB->width,frameRGB->height,24);
     ActualVideo=PreviewVideo.GetImage();

     // Enabling picture
     pictureRGB = cvLoadImage("SplineStatic.jpg",1);
     PreviewPicture.Create(pictureRGB->width,pictureRGB->height,24);
     ActualPicture = PreviewPicture.GetImage();

}


void CAUMERGUIDlg::OnTimer(UINT nIDEvent)
{
     switch (nIDEvent)
     {
     case 1:
          if(nav.GetNavigationMode() == MANUAL)
          {
               RefreshStaticVideo();
          }

          else if(nav.GetNavigationMode() == AUTO && nav.GetStartNavigationFcnFlag() == false)
          {
               nav.SetStartNavigationFcnFlag(true);
               frameRGB = cvQueryFrame(capture);
               if( !frameRGB ) MessageBox("Could not initialize Refresh capturing.");

               UpdateData(true);

               bool test;
```

```
                t                    e                    s                    t
=(nav.StartNavigation(atoi(m_PositionXValue),atoi(m_PositionYValue),atoi(m_TensionValue),
frameRGB));
                if(!test)
                {
                     m_mscomm.put_Output( nav.SendMessage232(MOTORENABLE,0));
                }
                nav.GetNavigationMetrics(m_Metrics);

                //m_RadiusValue.Format("%.2f", m_SliderDirCtrl.GetPos());
                nav.SetWheelSpeedRef(m_SliderSpeedCtrl.GetPos(),     m_Metrics[3],
m_SliderDirCtrl.GetRangeMax());
                //Update of speed display in dialog
                m_RadiusValue.Format("%.1f", nav.GetRadiusRc());
                //Update of speed display in dialog
                //m_SpeedValue.Format("%.1f", nav.GetSpeedVc());
                m_SliderDirCtrl.SetPos(nav.GetRadiusRcSliderPos());
                m_YawValue.Format("%.2f", m_Metrics[2] );
                m_PositionYActualValue.Format("%.1f", m_Metrics[1]);
                m_PositionXActualValue.Format("%.1f", m_Metrics[0]);
                UpdateData(false);

                //Sending motor speed
                m_mscomm.put_Output( nav.SendMessage232(SPEED) );

                CvFont pfont;
                cvCopy(frameRGB,ActualVideo);
                cvInitFont( &pfont, CV_FONT_HERSHEY_PLAIN,1, 1, 0, 1 , CV_AA );
                ActualVideo->origin = 1;

                CvPoint pt1 = cvPoint(m_Metrics[6], ActualVideo->height - m_Metrics[7]);
                CvPoint pt2 = cvPoint(m_Metrics[4], ActualVideo->height - m_Metrics[5]);
                cvLine( ActualVideo, pt1, pt2,CV_RGB(200,0,0),1, CV_AA, 0 );

                // Redraw GUI video display
                ActualVideo->origin = 1;
                cvPutText(  ActualVideo,  "TRACKING  ON",  cvPoint(  200,  15  ),  &pfont,
CV_RGB(0,255,0) );


                pictureRGB = cvLoadImage("SplineDynamic.jpg", 1);
                if( !pictureRGB )
                     MessageBox("Could not load path trajectory.");
                cvCopy(pictureRGB, ActualPicture);
                ActualPicture->origin = pictureRGB->origin;
                OnPaint();

                nav.SetStartNavigationFcnFlag(false);

          }

          break;

     case 100:       // check for status bar clock timer
          CTime t1 = 0;
          t1=CTime::GetCurrentTime();
          m_bar.SetPaneText(1,t1.Format("\t%H:%M:%S"));
          break;
     //default:
     //    break;
     }
     //Resume with generic implementation of the function
     CDialog::OnTimer(nIDEvent);
}
BOOL CAUMERGUIDlg::PreTranslateMessage(MSG* pMsg)
{
     // TODO: Add your specialized code here and/or call the base class


     if(pMsg->message==WM_KEYDOWN)
     {
          int SliderDirPos = m_SliderDirCtrl.GetPos();
          if(abs(SliderDirPos) == SliderDirPos)
                    SliderDirPos = (-SliderDirPos + m_SliderDirCtrl.GetRangeMax());
               else
                    SliderDirPos = -(SliderDirPos + m_SliderDirCtrl.GetRangeMax());


          switch(pMsg->wParam)
          {
          // Capturing change in speed command (manual mode)
          case VK_UP: // Up Arrow Pressed
              pMsg->wParam='A';      // Makes windows thinks it is an A.
              m_SliderSpeedCtrl.SetPos(m_SliderSpeedCtrl.GetPos() + 28);
               //Update display text box
               //m_SpeedValue.Format("%.2lf", m_SliderSpeedCtrl.GetPos());
               break;

          // Capturing change in speed command (manual mode)
          case VK_DOWN: // Down Arrow Pressed
               pMsg->wParam='A';     // Makes windows thinks it is an A.
               m_SliderSpeedCtrl.SetPos(m_SliderSpeedCtrl.GetPos() - 28);
               //Update display text box
               //m_SpeedValue.Format("%.2lf", m_SliderSpeedCtrl.GetPos());
               break;

          // Capturing change in steering command (manual mode)
          case VK_LEFT: // Left Arrow Pressed
              pMsg->wParam='A';      // Makes windows thinks it is an A.
               if(SliderDirPos == 1024)
                    SliderDirPos  = -SliderDirPos;
                    SliderDirPos += 32; //m_SliderDirCtrl.GetPos() ;

               //m_SliderDirCtrl.SetPos(m_SliderDirCtrl.GetPos() - 128);
               //Update display text box
               //m_RadiusValue.Format("%.2lf", m_SliderDirCtrl.GetPos());
               break;

          // Capturing change in steering command (manual mode)
          case VK_RIGHT:              // Right Arrow Pressed
              pMsg->wParam='A';      // Makes windows thinks it is an A.
              SliderDirPos -= 32;

               //m_SliderDirCtrl.SetPos(m_SliderDirCtrl.GetPos() + 128);
               //m_RadiusValue.Format("%.2lf", m_SliderDirCtrl.GetPos());
               break;

          case VK_INSERT:
               Beep( 320, 100 );
               break;
```

```
        default:
             break;
        }
        UpdateData(true);
        nav.SetWheelSpeedRef(m_SliderSpeedCtrl.GetPos(),SliderDirPos,m_SliderDirCtrl.GetRan
        geMax());
        //Update of speed display in dialog
        m_RadiusValue.Format("%.1lf", nav.GetRadiusRc());
        //Update of speed display in dialog
        m_SpeedValue.Format("%.0lf", nav.GetSpeedVc());
        m_SliderDirCtrl.SetPos(nav.GetRadiusRcSliderPos());
        UpdateData(false);

        m_mscomm.put_Output( nav.SendMessage232(SPEED,0,0) );

        /*   BYTE SpeedPacket[11] = {'<','S','0','0','0','0','0','0','0','0','>'};
        int j = 0;
        for (int i = 2;i < 6 ;i++)
        {
             SpeedPacket[i] = nav.wlByte[j];
             SpeedPacket[i+4] = nav.wrByte[j];
             j++;
        }
        CByteArray bArray;

        for(UINT n = 0; n < (sizeof(SpeedPacket) / sizeof(BYTE)); n++)
             bArray.Add(SpeedPacket[n]);

        COleVariant var = bArray;
        m_mscomm.put_Output(var);*/


        //nav.SendMessage(SPEED);
        //nav.SendMessage();//SPEED);
        //CString sOut;                    //Generate an output.
        //sOut = _T("Hello");              // Build your output...
        //CComVariant vOut(sOut);       // Generate a variant for the output.
        //m_mscomm.put_Output(vOut);    // Send the command.
    }
   return CDialog::PreTranslateMessage(pMsg);
}
void CAUMERGUIDlg::OnFileExit()
{
     PostMessage(WM_COMMAND, IDOK, 0L);
     // TODO: Add your command handler code here
}

void CAUMERGUIDlg::OnCameraSource()
{
     // TODO: Add your command handler code here
     cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_SOURCE, 0 );
}

void CAUMERGUIDlg::OnCameraDisplay()
{
     // TODO: Add your command handler code here
     cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_DISPLAY, 0 );
}

void CAUMERGUIDlg::OnCameraCompression()
{
     // TODO: Add your command handler code here
     cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_COMPRESSION, 0 );
}

void CAUMERGUIDlg::OnCameraFormat()
{
     // TODO: Add your command handler code here
     cvSetCaptureProperty(capture, CV_CAP_PROP_DIALOG_FORMAT, 0 );
}
void CAUMERGUIDlg::OnCommMscomm1()
{

     // TODO: Add your message handler code here
     if (m_mscomm.get_CommEvent() == 2 )
    {
          COleSafeArray svar = m_mscomm.get_Input();
          void* pSrc;
          BYTE byteArray[11] = {0};
          svar.AccessData( &pSrc );
          // typecast to byte
          LPBYTE pbtSrc = (LPBYTE)pSrc;
          memcpy(byteArray, pbtSrc, sizeof(byteArray));
          svar.UnaccessData();

          switch(byteArray[1])
          {
               case 'S':
               {
                    BYTE SpeedBuffer[2];
                    short MotorSpeedLeftActual = 0;
                    short MotorSpeedRightActual = 0;

                    SpeedBuffer[1] = byteArray[2];
                    SpeedBuffer[0] = byteArray[3];
                    memcpy(&MotorSpeedLeftActual,SpeedBuffer,sizeof(SpeedBuffer));

                    SpeedBuffer[1] = byteArray[4];
                    SpeedBuffer[0] = byteArray[5];
                    memcpy(&MotorSpeedRightActual,SpeedBuffer,sizeof(SpeedBuffer));
                    nav.SetSpeedVcActual(MotorSpeedLeftActual, MotorSpeedRightActual);
                    m_SpeedActualValue.Format("%.1lf", nav.GetSpeedVcActual());

                    //If navigation mode is automatic log actual motor speeds
                    if(nav.GetNavigationMode() == AUTO )
                         nav.SaveSpeedActual();
               return;
               }

               case 'P':
               {
                    BYTE PositionBuffer[4];
                    unsigned int MotorPositionLeftActual = 0;
                    unsigned int MotorPositionRightActual = 0;

                    PositionBuffer[3] = byteArray[2];
                    PositionBuffer[2] = byteArray[3];
                    PositionBuffer[1] = byteArray[4];
                    PositionBuffer[0] = byteArray[5];
                    memcpy(&MotorPositionRightActual,PositionBuffer,sizeof(PositionBuffer));

                    PositionBuffer[3] = byteArray[6];
```

```
                    PositionBuffer[2] = byteArray[7];
                    PositionBuffer[1] = byteArray[8];
                    PositionBuffer[0] = byteArray[9];

                    memcpy(&MotorPositionLeftActual,PositionBuffer,sizeof(PositionBuffer));
                    nav.SetPositionActual(MotorPositionLeftActual, MotorPositionRightActual);

                    //If navigation mode is automatic log actual motor position
                    if(nav.GetNavigationMode() == AUTO )
                         nav.SavePositionActual();

                    return;
               }
               case 'V':
               {
                    m_VoltageLevelCtl.SetPos((int)byteArray[2]);

                    return;
               }

               default:
               {
                    //Reset input buffer if message not recognized, possible buffer corruption
                    m_mscomm.put_InBufferCount(0);                    //Clear the input
buffer
                    return;
               }
          }
     }
}

void CAUMERGUIDlg::OnMouseMove(UINT nFlags, CPoint point)
{
     // TODO: Add your message handler code here and/or call default
    CString s;
    s.Format("     X=%d Y=%d",point.x,point.y);
    m_bar.SetPaneText(0,s);           //Update status bar with cursor position
    CDialog::OnMouseMove(nFlags, point);
}
void CAUMERGUIDlg::RefreshStaticVideo()
{

     frameRGB = cvQueryFrame(capture);
     if( !frameRGB )
          MessageBox("Could not initialize Video capturing.");
     cvCopy(frameRGB,ActualVideo);
     ActualVideo->origin = frameRGB->origin;
     //cvInitFont( &pfont, CV_FONT_HERSHEY_PLAIN,1, 1, 0, 1 , CV_AA );
     //cvPutText( ActualVideo, "TRACKING ON", cvPoint( 200, 15 ), &pfont, CV_RGB(0,255,0) );
     //cvLine( ActualVideo, cvPoint(118,0), cvPoint(65,240), CV_RGB(255,0,0),1, CV_AA, 0 );
     //cvCircle( ActualVideo, cvPoint(50,50),3,CV_RGB(255,0,0),-1, CV_AA,0);
     //cvPutText( ActualVideo, "+", cvPoint( 44, 45 ), &pfont, CV_RGB(255,0,0) );
     // Redraw GUI video display

     pictureRGB = cvLoadImage("SplineStatic.jpg", 1);
     if( !pictureRGB )
          MessageBox("Could not load path trajectory.");
     cvCopy(pictureRGB, ActualPicture);
     ActualPicture->origin = pictureRGB->origin;
     OnPaint();
}
void CAUMERGUIDlg::OnBnClickedStart()
{
     // TODO: Add your control notification handler code here
     //m_AutoButtonCtl.SetWindowText(_T("Loading..."));
     if(!MatlabDllInit)
     {
          m_StartButtonCtl.SetWindowText(_T("Loading..."));
          CWaitCursor wait;
          if(!(MatlabDllInit = libmatrixpInitialize()))
          {
            fprintf(stderr, "Could not initialize Matlab Dll.\n");
             exit(1);
          }
     }
     nav.SetNavigationMode(AUTO);
     m_mscomm.put_OutBufferCount(0);                    //Clear the output buffer
     m_mscomm.put_InBufferCount(0);                     //Clear the input buffer
     m_StartButtonCtl.SetWindowText(_T("Start"));
     m_StartButtonCtl.EnableWindow(FALSE);
     m_StopButtonCtl.EnableWindow(TRUE);
     m_StopButtonCtl.SetFocus();

}
void CAUMERGUIDlg::OnBnClickedButtonstop()
{
     // TODO: Add your control notification handler code here
     nav.SetNavigationMode(MANUAL);
     m_StopButtonCtl.EnableWindow(FALSE);
     m_StartButtonCtl.EnableWindow(TRUE);
     m_StartButtonCtl.SetFocus();
}
void CAUMERGUIDlg::OnBnClickedRadiodisable()
{
     // TODO: Add your control notification handler code here
     m_mscomm.put_Output( nav.SendMessage232(MOTORENABLE,0) );
}

void CAUMERGUIDlg::OnBnClickedRadioenable()
{
     // TODO: Add your control notification handler code here
     m_mscomm.put_Output( nav.SendMessage232(MOTORENABLE,1));
}
void CAboutDlg::OnPaint()
{
     CPaintDC dc(this); // device context for painting
     // TODO: Add your message handler code here
     // Do not call CDialog::OnPaint() for painting messages
}

void CAUMERGUIDlg::OnDeltaposSpinproportional(NMHDR *pNMHDR, LRESULT *pResult)
{
     //LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
     // TODO: Add your control notification handler code here
     UpdateData(true);
     NM_UPDOWN* spin = (NM_UPDOWN*)pNMHDR;
     m_mscomm.put_Output(     nav.SendMessage232(PIDGAIN,(*spin).iPos+(*spin).iDelta,
atoi(m_IntegralValue)) );
     *pResult = 0;
}
```

```
void CAUMERGUIDlg::OnDeltaposSpinintegral(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
	// TODO: Add your control notification handler code here
	UpdateData(true);
	NM_UPDOWN* spin = (NM_UPDOWN*)pNMHDR;
	m_mscomm.put_Output(	nav.SendMessage232(PIDGAIN,
atoi(m_ProportionalValue),(*spin).iPos+(*spin).iDelta ) );
	*pResult = 0;
}
void CAUMERGUIDlg::OnCancel()
{
	// TODO: Add your specialized code here and/or call the base class
	m_mscomm.put_Output( nav.SendMessage232(MOTORENABLE,0) ); // Disable motor before leaving
	CDialog::OnCancel();
}

//-----------------------------------------------------------------------------------------------
// CNavigation.cpp : implementation file
//-----------------------------------------------------------------------------------------------

#include ".\CNavigation.h"
//#include "PreciseTimer.h"

CNavigation::CNavigation(void)
{
	Rwheel = 3.4375;		//Radius of the wheels [inches]
	L = 16.25;			//Width of the cart
	wr = 0;
	wl = 0;
	Vc = 0;
	wrActual = 0;
	wlActual = 0;
	VcActual = 0;
	RcspSliderPos = 1024;
	Rc = 1024;
	pwl = &wl;
	pwr = &wr;
	pVc = &Vc;
	pRc = &Rc;
	direction = true;
	steering = false;
	NavigationMode = MANUAL;
	frameGRAY = cvCreateImage( cvSize( 320, 240 ), 8, 1 );
	ImgBuffer = NULL;
	OutputBuffer = NULL;
	StartNavigationFcnFlag = false;

	for(int i = 0; i < 10; i++)
		Metrics[i] = 0;

	for(int i = 0; i < (sizeof(wlByte)); i++)
	{
		wlByte[i] = '0';
		wrByte[i] = '0';
	}
}

CNavigation::~CNavigation(void)
{
}
```

```
void	CNavigation::SetWheelSpeedRef(int	SpeedSetPoint,	int	CurvatureRadiusSetPoint,	int
RadiusRangeMAX)
{
const float w_MAX = 255;
const float wl_MAX = w_MAX;
const float wr_MAX = w_MAX;
const bool	RIGHT = 1;
const bool	LEFT = 0;
const bool	REVERSE = 0;
const bool	FORWARD = 1;
const float Vcsp_MAX = w_MAX*Rwheel;

//Variable initialization
float wr = 0;				//Desired Actual right wheel speed
float wl = 0;				//Desired Actual left wheel speed
float Vc = 0;
float Rc = 0;

//Parameters pre-processing
float Vcsp = (float)SpeedSetPoint;	//Desired speed set point at center of robot

	if(Vcsp < 0)
		direction = REVERSE;
	else
		direction = FORWARD;

	if(fabs(Vcsp) > Vcsp_MAX)
		Vcsp = Vcsp_MAX;
	else
		Vcsp = fabs(Vcsp);

float Rcsp = (float)CurvatureRadiusSetPoint;

	if (Rcsp < 0)
		steering = LEFT;
	else
		steering = RIGHT;

	// Use for slider display
	if(fabs(Rcsp) == Rcsp)
		RcspSliderPos = (int)(-Rcsp + RadiusRangeMAX);//exp(-fabs(0.002*Rcsp));
	else
		RcspSliderPos = -(int)(Rcsp + RadiusRangeMAX);//exp(-fabs(0.002*Rcsp));

	Rcsp = fabs(Rcsp);

//Wheel speed computation
float X = (2*Rwheel*Rcsp)/(2*Rcsp+L);

	//For the case of a right turn
	if ( (Rcsp > 0) && (steering == RIGHT) )
	{
		wl = Vcsp/X;
		if(wl > wl_MAX)
		{
			wl = wl_MAX;
			Vc = X*wl;
			wr = (2/Rwheel)*Vc-wl;
		}
		else
		{
```

```
                wr = (2/Rwheel)*Vcsp-wl;
                Vc = Vcsp;
            }
            Rc = CurvatureRadiusSetPoint;
        }

        else if ( (Rcsp == 0) && (steering == RIGHT) )
        {
            wl = wl_MAX;
            wr = -wl;
            Vc = 0;
            Rc = CurvatureRadiusSetPoint;
        }

        //For the case of a left turn
        else if ( (Rcsp > 0) && (steering == LEFT) )
        {
            wr = Vcsp/X;
            if (wr > wr_MAX)
            {
                wr = wr_MAX;
                Vc = X*wr;
                wl = (2/Rwheel)*Vc-wr;
            }
            else
            {
                wl = (2/Rwheel)*Vcsp-wr;
                Vc = Vcsp;
            }
            Rc = CurvatureRadiusSetPoint;
        }

        else if ( (Rcsp == 0) && (steering == LEFT) )
        {
            wr = wr_MAX;
            wl = -wr;
            Vc = 0;
            Rc = CurvatureRadiusSetPoint;
        }

        //Updating desired motor speeds, speed of robot , radius of curvature
        if (direction == FORWARD)
        {
            *pwl = 255;//(int)floor(wl);
            *pwr = 255;//(int)ceil(wr);
            *pVc = Vc;
            *pRc = Rc;
        }

        else if (direction == REVERSE)
        {
            *pwl = -( (short)floor(wl));
            *pwr = -( (short)ceil(wr) );
            *pVc = -Vc;
            *pRc = Rc;
        }
}

float CNavigation::GetRadiusRc(void)
{
    return Rc;
}
int CNavigation::GetRadiusRcSliderPos(void)
{
    return RcspSliderPos;
}
float CNavigation::GetSpeedVc(void)
{
    return Vc;
}
COleVariant CNavigation::SendMessage232(rs232msg Message, int param1, int param2)
{
    CByteArray bArray;
    switch(Message)
    {
        case SPEED:
        {
            BYTE SpeedPacket[11] = {'<','S','0','0','0','0','0','0','0','0','>'};
            int j = 0;
            SpeedtoByte();
            for (int i = 2;i < 6 ;i++)
            {
                SpeedPacket[i] = wlByte[j];
                SpeedPacket[i+4] = wrByte[j];
                j++;
            }

            for(UINT n = 0; n < (sizeof(SpeedPacket) / sizeof(BYTE)); n++)
                bArray.Add(SpeedPacket[n]);

            COleVariant var = bArray;
            return var;
        }

        case PIDGAIN:
        {
            BYTE PIDGainPacket[11] = {'<','G','0','0','0','0','0','0','0','0','>'};
            int j = 0;
            Proportional = param1;
            Integral = param2;
            GaintoByte();
            for (int i = 2;i < 6 ;i++)
            {
                PIDGainPacket[i] = ProportionalByte[j];
                PIDGainPacket[i+4] = IntegralByte[j];
                j++;
            }

            for(UINT n = 0; n < (sizeof(PIDGainPacket) / sizeof(BYTE)); n++)
                bArray.Add(PIDGainPacket[n]);

            COleVariant var = bArray;
            return var;
        }

        case MOTORENABLE:
        {
            if ( (bool)param1 == false) {
                BYTE MotorEnablePacket[] = {'<','E','0','>'};

                for(UINT n = 0; n < (sizeof(MotorEnablePacket) / sizeof(BYTE)); n++)
                    bArray.Add(MotorEnablePacket[n]);
```

```
                }
                else if ( (bool)param1 == true) {
                     BYTE MotorEnablePacket[] = {'<','E','1','>'};

                     for(UINT n = 0; n < (sizeof(MotorEnablePacket) / sizeof(BYTE)); n++)
                          bArray.Add(MotorEnablePacket[n]);
                }

                COleVariant var = bArray;
                return var;
           }
           default:
                break;
     }



     //switch(Message)
     //{
     //    case SPEED:
     //         CString sOut;                    // Generate an output.
     //         sOut = _T("Hello world");  // Build your output...
     //         CComVariant vOut(sOut);          // Generate a variant for the output.
     //         m_mscomm.put_Output(vOut);       // Send the command.
     //         break;
     //}

   //UpdateData(TRUE);
   //m_comm.SetOutput(COleVariant(m_output));

}

void CNavigation::itob(short n, char s[])
{
     /*  Stores a string representation of integer n
    in s[], using a numerical base of b. Will handle
    up to base-36 before we run out of digits to use.  */

     static char digits[] = "0123456789ABCDEF";
    unsigned short d = 0;
     unsigned int i, j;
     int c, sign;

     if ((sign = n) < 0)
          d = (unsigned short)n;
     else
          d = n;

     i = 0;
     do
     {
          s[i++] = digits[d % 16];
     }
     while ((d /= 16) > 0);

     if ((sign = n) < 0)
          while (i < sizeof(s)) s[i++] = 'f';        //padding with fs
     else
          while (i < sizeof(s)) s[i++] = '0';        //padding with 0s
     s[i] = '\0';

     //Reverses string s[] in place
    for ( i = 0, j = sizeof(s)-1; i < j; i++, j--)
     {
          c = s[i];
       s[i] = s[j];
       s[j] = c;
     }
}

void CNavigation::SpeedtoByte(void)
{
     itob(wl, wlByte);
     itob(wr, wrByte);
}
void CNavigation::GaintoByte(void)
{
     itob(Integral, IntegralByte);
     itob(Proportional, ProportionalByte);
}
/*   CString sMessage = TEXT("");
     sMessage.Format(TEXT("wr: %d, wl: %d, Rcsp: %f,Vc: %f"),*pwr,*pwl,Rcsp, Vc);
     AfxMessageBox(sMessage, MB_OK|MB_ICONINFORMATION);
*/
bool CNavigation::SetNavigationMode(navmode mode)
{
     NavigationMode = mode;
     if(NavigationMode == AUTO)
     {
          // Delete the content of the navigation metric log files and and initializing headers
          datafile.open("DataLogMSVC.txt", ios::out);
          datafile.precision(4);

          datafile   <<   "time "                  <<"\t"\
                     <<   "Cx"                     <<"\t"\
                     <<   "Cy"                     <<"\t"\
                     <<   "theta"                       <<"\t"\
                     <<   "Radius"                 <<"\t"\
                     <<   "vpx"                         <<"\t"\
                     <<   "Vpy"                         <<"\t"\
                     <<   "Cz"                     <<"\t"\
                     <<   "phi"                         <<"\n";

          datafile.close();

          //Initializing header in log file of motor speeds
          datafile.open("SpeedLogMSVC.txt", ios::out);
          datafile.precision(4);

          datafile   <<   "time"                   <<"\t"\
                     <<   "wlActual"                    <<"\t"\
                     <<   "wl"                     <<"\t"\
                     <<   "wrActual"                    <<"\t"\
                     <<   "wr"                     <<"\t"\
                     <<   "VcActual"                    <<"\t"\
                     <<   "Vc"                     <<"\n";

          datafile.close();

          //Initializing header in log file of motor position
          datafile.open("PositionLogMSVC.txt", ios::out);
```

```
            datafile   <<    "timestamp"                <<"\t"\
                             <<    "positionlActual"     <<"\t"\
                             <<    "positionrActual"     <<"\n";

            datafile.close();

            // Start timer
            TickCount.StartTimer();
      }
      else
            TickCount.StopTimer();

      return true;
}

int CNavigation::GetNavigationMode(void)
{
      return NavigationMode;
}

bool CNavigation::StartNavigation(int Cx, int Cy, int S, IplImage* image)
{
      frameRGB = image;
      // Converting image to grayscale
      cvCvtColor( frameRGB, frameGRAY, CV_RGB2GRAY );
      frameGRAY->origin = frameRGB->origin;

      // Extracting data from Ipl image structure to 1D buffer
      ImgBuffer = new /*(std::nothrow)*/ mxUint8[frameGRAY->height*frameGRAY->width];
      if (ImgBuffer == 0)
            AfxMessageBox("Error: memory could not be allocated");

      int z = 0;//frameGRAY->width*frameGRAY->height - 1;
      int x = 0;
      int y = frameGRAY->height - 1;
      while ( x <= frameGRAY->width - 1){
            while (y >= 0){
                  ImgBuffer[z] = ((uchar*)(frameGRAY->imageData + frameGRAY->widthStep*y))[x];
                  z++;
                  y--;
            }
            y = frameGRAY->height - 1;
            x++;
      }


      mxDouble ParamBuffer[] = {Cx,Cy,S};

      try  {

            mwArray MatlabOut(1,10,mxDOUBLE_CLASS, mxREAL);
            mwArray MatlabParamIn(1,3,mxDOUBLE_CLASS, mxREAL);
            MatlabParamIn.SetData(ParamBuffer,3);

            mwArray MatlabImageIn(frameGRAY->height,frameGRAY->width, mxUINT8_CLASS, mxREAL);
            //Transferring buffer to matlab array
            MatlabImageIn.SetData(ImgBuffer,frameGRAY->width*frameGRAY->height);
            //// Processing image array with Matlab compiled library

            homography(1,MatlabOut, MatlabParamIn, MatlabImageIn);
            nElements = MatlabOut.NumberOfElements();

            OutputBuffer = new (std::nothrow) float[nElements];

            //if (OutputBuffer == 0)
            //   AfxMessageBox("Error: memory could not be allocated");
            //
            MatlabOut.GetData(OutputBuffer,nElements);
            memcpy(Metrics,OutputBuffer,nElements*sizeof(OutputBuffer));
            SaveNavigationMetrics();
            delete[] ImgBuffer;
            delete[] OutputBuffer;
            return true;
      }

      catch (const mwException& e)
      {
            AfxMessageBox(e.what());
            return false;
      }

      catch (...)
      {
            AfxMessageBox("Unexpected error thrown.");
            return false;
      }

//    cvCopy(frameRGB,actual);
//    actual->origin = frameRGB->origin;
      // Redraw GUI video display
      //OnPaint();
}
void CNavigation::SetStartNavigationFcnFlag(bool value)
{
      StartNavigationFcnFlag = value;
}

bool CNavigation::GetStartNavigationFcnFlag()
{
      return StartNavigationFcnFlag;
}

void CNavigation::GetNavigationMetrics(float* destVector)
{
      for (int i = 0; i < 8; i++)
            destVector[i] = Metrics[i];
}
void CNavigation::SaveNavigationMetrics()
{
      __int64 timestamp = TickCount.GetTime();
      datafile.open("DataLogMSVC.txt", ios::app);

      datafile   <<   timestamp  <<"\t"\
                      <<   Metrics[0]      <<"\t"\
                      <<   Metrics[1]      <<"\t"\
                      <<   Metrics[2]      <<"\t"\
                      <<   Metrics[3]      <<"\t"\
                      <<   Metrics[6]      <<"\t"\
                      <<   Metrics[7]      <<"\t"\
                      <<   Metrics[8]      <<"\t"\
                      <<   Metrics[9]      <<"\n";

      datafile.close();
```

```
}
void CNavigation::SetSpeedVcActual(short SpeedLeftActual, short SpeedRightActual )
{
     wlActual = SpeedLeftActual;
     wrActual = SpeedRightActual;
     VcActual = Rwheel*(SpeedLeftActual + SpeedRightActual)/2;
}

void   CNavigation::SetPositionActual(unsigned   int   PositionLeftActual,   unsigned   int
PositionRightActual )
{
     positionlActual = PositionLeftActual;
     positionrActual = PositionRightActual;
}
float CNavigation::GetSpeedVcActual()
{
     return VcActual;
}
void CNavigation::SaveSpeedActual()
{
     __int64 timestamp = TickCount.GetTime();
     datafile.open("SpeedLogMSVC.txt", ios::app);

     datafile   <<   timestamp  <<"\t"\
                 <<   wlActual   <<"\t"\
                 <<   wl              <<"\t"\
                 <<   wrActual   <<"\t"\
                 <<   wr              <<"\t"\
                 <<   VcActual   <<"\t"\
                 <<   Vc              <<"\n";

     datafile.close();
}
void CNavigation::SavePositionActual()
{
     __int64 timestamp = TickCount.GetTime();
     datafile.open("PositionLogMSVC.txt", ios::app);

     datafile   <<   timestamp       <<"\t"\
                 <<   positionlActual <<"\t"\
                 <<   positionrActual <<"\n";

     datafile.close();
}

CString CNavigation::GetElapsedTime()
{
          CString ticks;
          CString timeH;
          CString timeL;
          __int64 aa = TickCount.GetTime();
          ticks.Format("%I64d", aa);
          timeL = ticks.Right(6);
          ticks.Delete(ticks.GetLength()-6,6);
          timeH = ticks;
          ElapsedTime = timeH + '.' + timeL + " sec";
          AfxMessageBox(ElapsedTime);
          int a = 1;
          return ElapsedTime;
}
```

C.3 High-Level Navigational Algorithm

```
;-------------------------------------------------------------------------------------------
;File name:              Homography.m
;
;Description:           Visual pose estimation and pose tracking controller.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2006
;
;-------------------------------------------------------------------------------------------

function m = Homography(Param, I)

    % Initialization
    psi   = ( 0 )*deg2rad;
    theta = ( 0 )*deg2rad;
    psi   = ( 0 )*deg2rad;
    Cx_desired =  Param(1);          % Desired lateral position;
    Cy_desired =  Param(2); %110;    % Look ahead distance
    s_desired  =  Param(3); %10;     % Shape parameter of the spline
    Cy_desired = -Cy_desired;        % Sign change for compatibility with GUI
    Cy = -48;                        % Actual longitudinal position
    Cz = -79;                        % Height of ceiling (reference metric)

    % Initialisation of matrices
    H=[]; Peaks=[]; R=[]; beta=[]; rho=[]; MB=[]; M=[]; B=[]; VP=[];

  %-----------------------------------------------------------------------------------------
    % Camera Internals
  %-----------------------------------------------------------------------------------------
    alpha_x = 400;                          % Focal length in x direction
    alpha_y = 400;                          % Focal length in y direction
    skew = 0;                               % Skew coefficient
    x_o = 180;                              % Principal point x-offset
    y_o = 120;                              % Principal point y-offset

    % Calibration matrix
    K = [alpha_x skew x_o; 0 (-alpha_y) y_o; 0 0 1];
    K2 = K*[-1 0 0; 0 -1 0; 0 0 1];

  %-----------------------------------------------------------------------------------------
    % Image Processing
  %-----------------------------------------------------------------------------------------
    % Binerization
    IBW = im2bw(I,.55);                 % Black feature on black background
    IWB = (~IBW);                       % White feature on black background

    % Extracting feature lines
    [H, TT, RR] = hough(IWB);
    Peaks  = houghpeaks(H, 6,'threshold', ceil(0.2*max(H(:))),'NHoodSize',[39 15]);
    beta = TT(Peaks(:,2))';
    rho  = RR(Peaks(:,1))';

    % Transformation of the parameters
    MB = [tan(pi/180*(90-beta)), rho./cos(pi/180*(90-beta))];

    % Classification of the vector M with respect to the
    % slopes of the extracted line features
    MB = sortrows(MB,1);

    % Search good line feature candidates
    j = 1;
    for i = 1:length(MB(:,1))
        if (MB(i,1) > 1.25 || MB(i,1) < -1.25)
            M(j,1) = MB(i,1);
            B(j,1) = MB(i,2);
            j = j+1;
        end
    end
  %-----------------------------------------------------------------------------------------
    % Vanishing Point Estraction
  %-----------------------------------------------------------------------------------------
    % Finding the vanishing point
    if (length(M(:,1)) > 0)
        a = 0; k = 1; z = 1;
        for i = 1:length(M(:,1))
            for j = 1:length(M(:,1))
                if (j < i)
                    x_int = ( B(j) - B(i) )/( M(j) - M(i) );
                    if (x_int >= 0 && x_int <= 640)
                        y_int = -M(i)*x_int + B(i);
                        if (y_int >= 0 && y_int <= 240)
                            INT(k,:) = [x_int y_int];
                            k = k + 1;
                            else if (y_int > 240)
                                % Save vanishing point candidates
                                VP(z,:) = [x_int y_int];
                                z = z + 1;
                            end
                        end
                    end
                end
            end
        end

    % Check if a vanishing point has been identified
    if ~isempty(VP)
        sumx_sq = 0; sumy_sq = 0;
        for i = 1:length(VP(:,1))
            sumx_sq = sumx_sq + VP(i,1)^2;
            sumy_sq = sumy_sq + VP(i,2)^2;
        end
        % Computation of the vanishing point (least square average)
        vpx = (sumx_sq/length(VP(:,1)))^0.5;
        vpy = (sumy_sq/length(VP(:,1)))^0.5;
    end
```

```
%-------------------------------------------------------------------------------------------
 % Projection Matrix
%-------------------------------------------------------------------------------------------
 % Recovery of the heading angle from vanishing point
 psi = atan2( alpha_y, -(vpy-y_o) );
 dpsi = 180/pi*psi;

 % Recovering the tilt angle from the vanishing point
 theta = atan2( (vpx-x_o)*sin(psi), alpha_x );
 dtheta = 180/pi*theta;

 % Recovery of translation vector
 Rx = [1 0 0; 0 cos(psi) sin(psi); 0 -sin(psi) cos(psi)];
 Rz = [cos(theta) sin(theta) 0; -sin(theta) cos(theta) 0; 0 0 1];
 R  = Rx*Rz;

 F = -K2*R;
 f11 = F(1,1); f12 = F(1,2); f13 = F(1,3);
 f21 = F(2,1); f22 = F(2,2); f23 = F(2,3);
 f31 = F(3,1); f32 = F(3,2); f33 = F(3,3);

 slope = -M(1);
 intercept = B(1);
 AA = (f12*Cy + f13*Cz);
 BB = (f22*Cy + f23*Cz);
 CC = (f32*Cy + f33*Cz);
 Cx = (slope*AA - BB + intercept*CC)/(-slope*f11 + f21 - intercept*f31);

 % Coordinate of desired ficticious target (landmark) on the image plane
 T = [Cx Cy Cz]';
 P = [K2*R -K2*R*T];
 GG = P*[Cx_desired 0 0 1]';
 xid = GG(1)/GG(3);
 yid = GG(2)/GG(3);

%-------------------------------------------------------------------------------------------
 % Pose Tracking Controller
%-------------------------------------------------------------------------------------------
 % Slope condition at spline end points
 theta0 = (theta + pi/2);
 theta1 = pi/2;

 % Control Points
 P1  = [Cx_desired Cy_desired];
 P0  = [Cx Cy];
 P_1 = P1 - [cos(theta0) sin(theta0)]/norm([cos(theta0), sin(theta0)]);
 P2  = P0 + [cos(theta1) sin(theta1)]/norm([cos(theta1), sin(theta1)]);
 Gc =[P_1; P0; P1; P2];

 % Cardinal spline
 [r, S, C] = CardinalSpline(s_desired, P_1, P0, P1, P2);

 % Maximum allowable radius of curvature
 if (r(1) < -1024)
     radius = -1024;
 elseif (r(1) > 1024)
     radius = 1024;
 else
     radius = r(1);
 end
```

```
%-------------------------------------------------------------------------------------------
 % Localization map (Human-Machine Interface)
%-------------------------------------------------------------------------------------------
 figure1 = figure(1);
 figure1 = clf('reset');
 set(figure1,'Visible','off','PaperUnits', 'points','PaperPosition',...
       [100 100 222 296],'Units','points','Position',[100 100 222 296]);
 box on; hold on; grid; axis equal; ax1 = gca;
 ylimits = [-80 48];
 xlimits = [-16 64];
 xinc = 8;
 yinc = 8;
 set(ax1,'XTick',[xlimits(1):xinc:xlimits(2)],...
                                    'YTick',[ylimits(1):yinc:ylimits(2)]);
 axis([xlimits(1),xlimits(2), ylimits(1),ylimits(2)]);
 xlabel('Lateral Position (in.)');
 ylabel('Longitudinal Position (in.)');
 title('Trajectory Tracking','FontWeight','light');


 % Plot of the desired path
 line([Cx_desired Cx_desired],[ylimits(2) ylimits(1)],...
                              'Linestyle','-','Color','r','Linewidth',2);

 % Drawing the robot icon
 carty = [-7 -7 7 7 -7];
 cartx = [4 -10 -10 4 4];
 Cart = [cartx; carty];

 wheelrighty = [-10 -10 -8 -8 -10];
 wheelrightx = [4 -4 -4 4 4];
 WheelRight = [wheelrightx; wheelrighty];

 wheellefty = [8 8 10 10 8];
 wheelleftx = [4 -4 -4 4 4];
 WheelLeft =[wheelleftx; wheellefty];

 Rot = [cos(theta0) -sin(theta0); sin(theta0) cos(theta0)];
 Trans = [Cx Cy]';
 for k = 1:1:4
     WR(:,k) = Rot*WheelRight(:,k) + Trans;
     WL(:,k) = Rot*WheelLeft(:,k) + Trans;
     CRT(:,k) = Rot*Cart(:,k) + Trans;
 end
 patch(WL(1,:),WL(2,:),[1 1 1],'linewidth', 1);
 patch(WR(1,:),WR(2,:),[1 1 1],'linewidth', 1);
 patch(CRT(1,:),CRT(2,:),[1 1 1],'FaceAlpha',0,'linewidth', 1);

 % Rendering the spline (robot path)
  plot(S(:,1),S(:,2),'-','color','k','linewidth', 1.5);
  plot(Gc(2,1),Gc(2,2),'s','color','b','linewidth',1);
  plot(Gc(3,1),Gc(3,2),'s','color','b','linewidth',1);

 % Plot the World Coordinate frame
 text(-5,18,'X_w','FontName','times new roman','FontSize',10,...
                                    'HorizontalAlignment','center');
 text(18,-5,'Y_w','FontName','times new roman','FontSize',10,...
                                    'HorizontalAlignment','center');
 patch([-2 0 2], [12 16 12],[0 0 0],'LineStyle','none','linewidth', 1);
 patch([12 16 12], [-2 0 2],[0 0 0],'LineStyle','none','linewidth', 1);
 plot(0,0,'k.','markersize',15)
```

```
    line([0 0 0 16],[0 16 0 0],'Linestyle','-','Color','k','Linewidth',1.5);

    % Print to file the localization map and actual pose of the robot
    print(figure1,'-r0','-djpeg', 'SplineDynamic');

    % Output of the function (The minus sign assure the compatibility
    % with the Human-Machine interface)
    m = [Cx Cy dtheta -radius xid yid vpx vpy Cz dpsi];
end

;--------------------------------------------------------------------------------------------
;File name:              CardinalSpline.m
;
;Description:           Compute a cardinal spline between the actual pose
;                       and desired configuration.
;
;Author:                D. Gagné-Roussel, University of Ottawa
;
;Date:                  January 2006
;
;--------------------------------------------------------------------------------------------

function [r, R, C] = CardinalSpline(s, P_1, P0, P1, P2);

     % Cubic cardinal matrix
     Mc = [0 1 0 0; 0 0 1 0; -s 0 s 0; 0 -s 0 s];

     % Cubic hermite matrix
     Mh = [2 -2 1 1; -3 3 -2 -1; 0 0 1 0; 1 0 0 0];

     % Cardinal geometry vector
     Gc =[P_1; P0; P1; P2];

     % Curve points
     i = 1;
     for u = 0:0.01:1
          % Position
         R(i,:) = [u^3, u^2, u, 1]*Mh*Mc*Gc;

         % First derivative
         dR(i,:) = [3*u^2, 2*u, 1, 0]*Mh*Mc*Gc;

         % Second derivative
         d2R(i,:) = [6*u, 2, 0, 0]*Mh*Mc*Gc;
         i = i + 1;
     end

     % Normal and Tangent vector
     i = 1;
     for u = 0:0.01:1
         % Position
         R(i,:) = [u^3, u^2, u, 1]*Mh*Mc*Gc;

         % First derivative
         dRdt(i,:) = [3*u^2 2*u 1 0]*Mh*Mc*Gc;
         dxdt = dRdt(i,1);
         dydt = dRdt(i,2);

         % Second derivative
         d2Rdt2(i,:) = [6*u 2 0, 0]*Mh*Mc*Gc;
         d2xdt2 = d2Rdt2(i,1);
         d2ydt2 = d2Rdt2(i,2);

         % Curvature k
         k(i) = (dxdt*d2ydt2 - dydt*d2xdt2)/norm(dRdt(i,:))^3;

         % Unit tangent
         T(i,:) = dRdt(i,:)/norm(dRdt(i,:));

         % Unit normal vector
         dTdt(i,:) = (norm(dRdt(i,:))*d2Rdt2(i,:) - ( (dRdt(i,1)*d2Rdt2(i,1)+...
                     dRdt(i,2)*d2Rdt2(i,2))/norm(dRdt(i,:)) )*dRdt(i,:) )/norm(dRdt(i,:))^2;
         N(i,:) = dTdt(i,:)/norm(dTdt(i,:));

         % Radius of curvature
         r(i) = 1/k(i);

         % Center of osculating circle
         C(i,:) = R(i,:) + r(i)*N(i,:);
         i = i+1;
     end
```

```
//--------------------------------------------------------------------------------------------
// MATLAB Compiler: 4.1 (R14SP1)
// Date: Mon May 22 20:15:05 2006
// Arguments: "-B" "macro_default" "-W" "cpplib:libmatrixp" "-T" "link:lib"
// "Homography.m" "CardinalSpline.m"
//--------------------------------------------------------------------------------------------

#ifndef __libmatrixp_h
#define __libmatrixp_h 1

#if defined(__cplusplus) && !defined(mclmcr_h) && defined(__linux__)
#  pragma implementation "mclmcr.h"
#endif
#include "mclmcr.h"
#include "mclcppclass.h"
#ifdef __cplusplus
extern "C" {
#endif

extern bool libmatrixpInitializeWithHandlers(mclOutputHandlerFcn error_handler,
                                             mclOutputHandlerFcn print_handler);
extern bool libmatrixpInitialize(void);
extern void libmatrixpTerminate(void);


extern void mlxHomography(int nlhs, mxArray *plhs[],
                          int nrhs, mxArray *prhs[]);

extern void mlxCardinalspline(int nlhs, mxArray *plhs[],
                              int nrhs, mxArray *prhs[]);

#ifdef __cplusplus
}
#endif

#ifdef __cplusplus

extern void homography(int nargout, mwArray& m
                       , const mwArray& Param, const mwArray& I);
```

```
extern void cardinalspline(int nargout, mwArray& r, mwArray& R
                           , mwArray& C, const mwArray& s
                           , const mwArray& P_1, const mwArray& P0
                           , const mwArray& P1, const mwArray& P2);

#endif

#endif

//--------------------------------------------------------------------------------------------
// MATLAB Compiler: 4.1 (R14SP1)
// Date: Mon May 22 20:15:05 2006
// Arguments: "-B" "macro_default" "-W" "cpplib:libmatrixp" "-T" "link:lib"
// "Homography.m" "CardinalSpline.m"
//--------------------------------------------------------------------------------------------
#include <stdio.h>
#include "libmatrixp.h"
#ifdef __cplusplus
extern "C" {
#endif
extern const unsigned char __MCC_libmatrixp_public_data[];
extern const char *__MCC_libmatrixp_name_data;
extern const char *__MCC_libmatrixp_root_data;
extern const unsigned char __MCC_libmatrixp_session_data[];
extern const char *__MCC_libmatrixp_matlabpath_data[];
extern const int __MCC_libmatrixp_matlabpath_data_count;
extern const char *__MCC_libmatrixp_classpath_data[];
extern const int __MCC_libmatrixp_classpath_data_count;
extern const char *__MCC_libmatrixp_mcr_runtime_options[];
extern const int __MCC_libmatrixp_mcr_runtime_option_count;
extern const char *__MCC_libmatrixp_mcr_application_options[];
extern const int __MCC_libmatrixp_mcr_application_option_count;
#ifdef __cplusplus
}
#endif

static HMCRINSTANCE _mcr_inst = NULL;

#if defined( _MSC_VER) || defined(__BORLANDC__) || defined(__WATCOMC__) || defined(__LCC__)
#include <windows.h>

static char path_to_dll[_MAX_PATH];

BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, void *pv)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        char szDllPath[_MAX_PATH];
        char szDir[_MAX_DIR];
        if (GetModuleFileName(hInstance, szDllPath, _MAX_PATH) > 0)
        {
             _splitpath(szDllPath, path_to_dll, szDir, NULL, NULL);
            strcat(path_to_dll, szDir);
        }
     else return FALSE;
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
    }
    return TRUE;
}
#endif
static int mclDefaultPrintHandler(const char *s)
{
    return fwrite(s, sizeof(char), strlen(s), stdout);
}

static int mclDefaultErrorHandler(const char *s)
{
    int written = 0, len = 0;
    len = strlen(s);
    written = fwrite(s, sizeof(char), len, stderr);
    if (len > 0 && s[ len-1 ] != '\n')
        written += fwrite("\n", sizeof(char), 1, stderr);
    return written;
}

bool libmatrixpInitializeWithHandlers(
    mclOutputHandlerFcn error_handler,
    mclOutputHandlerFcn print_handler
)
{
    if (_mcr_inst != NULL)
        return true;
    if (!mclmcrInitialize())
        return false;
    if (!mclInitializeComponentInstance(&_mcr_inst,
                                        __MCC_libmatrixp_public_data,
                                        __MCC_libmatrixp_name_data,
                                        __MCC_libmatrixp_root_data,
                                        __MCC_libmatrixp_session_data,
                                        __MCC_libmatrixp_matlabpath_data,
                                        __MCC_libmatrixp_matlabpath_data_count,
                                        __MCC_libmatrixp_classpath_data,
                                        __MCC_libmatrixp_classpath_data_count,
                                        __MCC_libmatrixp_mcr_runtime_options,
                                        __MCC_libmatrixp_mcr_runtime_option_count,
                                        true, NoObjectType, LibTarget,
                                        path_to_dll, error_handler,
                                        print_handler))
        return false;
    return true;
}

bool libmatrixpInitialize(void)
{
    return libmatrixpInitializeWithHandlers(mclDefaultErrorHandler,mclDefaultPrintHandler);
}

void libmatrixpTerminate(void)
{
    if (_mcr_inst != NULL)  mclTerminateInstance(&_mcr_inst);
}

void mlxHomography(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
{
    mclFeval(_mcr_inst, "homography", nlhs, plhs, nrhs, prhs);
}

void mlxCardinalspline(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
{
    mclFeval(_mcr_inst, "cardinalspline", nlhs, plhs, nrhs, prhs);
```

```
}

void homography(int nargout, mwArray& m, const mwArray& Param, const mwArray& I)
{
    mclcppMlfFeval(_mcr_inst, "homography", nargout, 1, 2, &m, &Param, &I);
}

void cardinalspline(int nargout, mwArray& r, mwArray& R, mwArray& C
                    , const mwArray& s, const mwArray& P_1, const mwArray& P0
                    , const mwArray& P1, const mwArray& P2)
{
    mclcppMlfFeval(_mcr_inst, "cardinalspline", nargout, 3,
                   5, &r, &R, &C, &s, &P_1, &P0, &P1, &P2);
}
}
```