AN INTEGRATED ENVIRONMENT FOR SIMULATION AND CONTROL OF MOBILE ROBOTS

By

RON BERKOWITZ

A thesis submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Mechanical Engineering

Written under the direction of

Peng Song

and approved by

_____

_____

_____

New Brunswick, New Jersey

January, 2011

ABSTRACT OF THE THESIS

An Integrated Environment for Simulation and Control of Mobile Robots

By RON BERKOWITZ

Thesis Director:
Peng Song

There have been increased interests in mobile robot research due to its many applications in areas such as material handling, explorations in hazardous environments, and military missions under extreme conditions. Many control schemes and robot systems have been developed, yet most of these systems eventually become individual experiments that are unique or specific to particular applications. It is very difficult to verify or reuse the controls developed and build upon the existing knowledge. We argue that it is necessary to develop an integrated experiment and simulation environment equipped with a user-friendly interface to examine existing controls and eventually serve as an experimental testbed for mobile robot research.

The main contributions of this thesis are the design and integration of a hardware in the loop simulation environment for mobile robot control and navigation. We developed an easy to use graphical user interface (GUI) that can provide the users with the overall access to various robot functions including sensor feedback, object recognition, and tools for implementing the control strategies to study robot behaviors.

ACKNOWLEDGEMENTS

There are a number of individuals to whom I owe thanks for assistance in completing this work. Prof. Peng Song, my advisor, has been very patient and always willing to take time to go over any issues and provide guidance and technical insights. I would like to thank my thesis committee members, Prof. Haim Baruh and Prof. Kimberly Cook-Chennault, and the Graduate Program Director, Prof. William J. Bottega, for reviewing this thesis and their helpful advices along my pursuit. I would also like to thank Mr. Ke Xu for his willingness to set aside time to both discuss and help troubleshoot my research.

Table of Contents

Chapter 1

Introduction

Due to the broad capabilities of mobile robot systems and the proliferation of computing capacity, extensive amount of research has been done in the past two decades in control and coordination of multiple mobile robots. The main reason of the rapid development in mobile robots is that they can help to reduce the demand of human presence for tasks in extreme environments. One of the major challenges in this area is for the robots to work in a fully autonomous way to achieve the goals specified by a person. Many control strategies have been reported and applied in a wide variety of applications, such as large-size object transportation [1], unknown environment explorations [2], map-making [3], landmine detection [4] and search-and-rescue missions [5, 6]. A good review on the applications of multi-robot systems can be found in [7].

The problems considered in this research are rooted from the fundamental areas of mobile robots - the planning and navigation of mobile robots, which answers to the basic question of moving from one location to another and achieve that safely (without collisions to the environment). Among different types of robots, wheeled mobile robots are considered closest to our daily life. Wheeled robots can have different topologies depending on the number and the configuration of wheels. Even though three wheels are sufficient for a robot to balance on the ground, additional wheels can still be added, but more complicated mechanisms need to be applied to keep all wheels on the ground if the

surface is not flat. A few representative examples of three-wheeled mobile robots with differential drives are Pioneer™ of ActivMedia Robotics, the robotic vacuum cleaner Roomba™ by iRobot and the general purpose robot ER1 from Evolution Robotics. The differential drives are a two-wheeled driving system with two independent driving motors. A comprehensive review of different types of mobile robots in academia and industry can be found in [8].

The robot used in this thesis is the ER1 mobile platform. The ER1 is a differential drive mobile robot which can be easily assembled and customized. The software provided allows for a quick installation of the needed drivers to control the robot's motion. The robot we assembled for this thesis has a passive front steering wheel, two differential driven rear wheels, and is equipped with an on-board webcam as its vision sensor.

1.1 Scope of the Thesis

With the recent increase in computing power available, there has been a great deal of research being done in the design of robot control schemes, however there is still limited work focusing on the development of an integrated simulation and testing environment that can readily examine and eventually further develop the existing controls.

This thesis focuses on the implementation of sensor-based robot control schemes and the design of an integrated simulation environment for sensor based robot navigation. The simulation environment allows for the robot to detect and track an object via user

specified control schemes and provide a general hardware-in-the-loop simulation and testing environment for mobile robot research.

In order to develop the integrated simulation environment for sensor based navigation, several challenges need to be overcome. The robot must be able to correctly identify a 2-D object with known geometry in a cluttered environment and calculate its extrinsic parameters such as its position and orientation. Next, the robot needs to plan the proper motion required to maintain a specified orientation and distance from the object. After determining the motion plans, the robot should be able to localize itself and follow the planned trajectory.

The main contributions of this thesis are the design and integration of a hardware in the loop simulation environment for mobile robot control and navigation. We created a user friendly graphical interface that can provide the overall access to various robot functions including sensor feedback, object recognition, and tools to implement control strategies and observe robot behaviors. The interface is developed on top of the Application Program Interface of the ER1 robot for communication and control. The object recognition is performed through the on-board camera. A toolbox has been built and integrated into the environment for camera calibration and color-based object identification.

1.2 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 presents details of the nonholonomic control schemes used in the experiments. Chapter 3 provides information

on the method and algorithms for camera calibration and experimental results for vision-based sensing. Chapter 4 studies the advantages and limitations of the hardware platform of the ER1 robot with experiments carried out on the original robot control center (RCC). Chapter 5 contains details of the object identification algorithm, vision-based tracking, teleoperation, and motion control simulations and experimental results. Chapter 6 describes the design and user interface for the integrated simulation and experimental environment and provides experimental results for the user interface implementation. Chapter 7 presents conclusions and suggests directions for future research.

Chapter 2

Mobile Robot Control

In this chapter, we present the kinematics, dynamics, and trajectory tracking control of a differential drive mobile robot. We describe an *integrator backstepping* method for robot motion control. The control input is determined by tracking a reference trajectory via an auxiliary control velocity $\mathbf{v_c}$. We demonstrate the effectiveness of the control method with the results from simulations for various reference trajectories.

2.1 Kinematics

Consider a differential drive, three-wheeled mobile robot as shown in Fig. 2.1, where the two rear wheels are driven by two independently controlled motors and the front wheel is passive. The velocity, **v**, of the robot, can be expressed as a vector consisting of the linear velocity of the robot, v, at a reference point C and the angular velocity, ω, around the vertical axis. In general, the velocity of the robot can be expressed as a function of the velocity of the driving wheels,

$$\mathbf{v} = \begin{bmatrix} v \\ \omega \end{bmatrix} = f(\omega_{p1}, \omega_{p2}), \tag{2.1}$$

where ω_{p1} and ω_{p2} are the angular velocities of the individual wheels. Assuming pure rolling between the wheels and the ground, for the robot shown in Fig. 2.1, **v** can be expressed as

$$\mathbf{v} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ r/R & -r/R \end{bmatrix} \begin{bmatrix} \omega_{p1} \\ \omega_{p2} \end{bmatrix}. \tag{2.2}$$

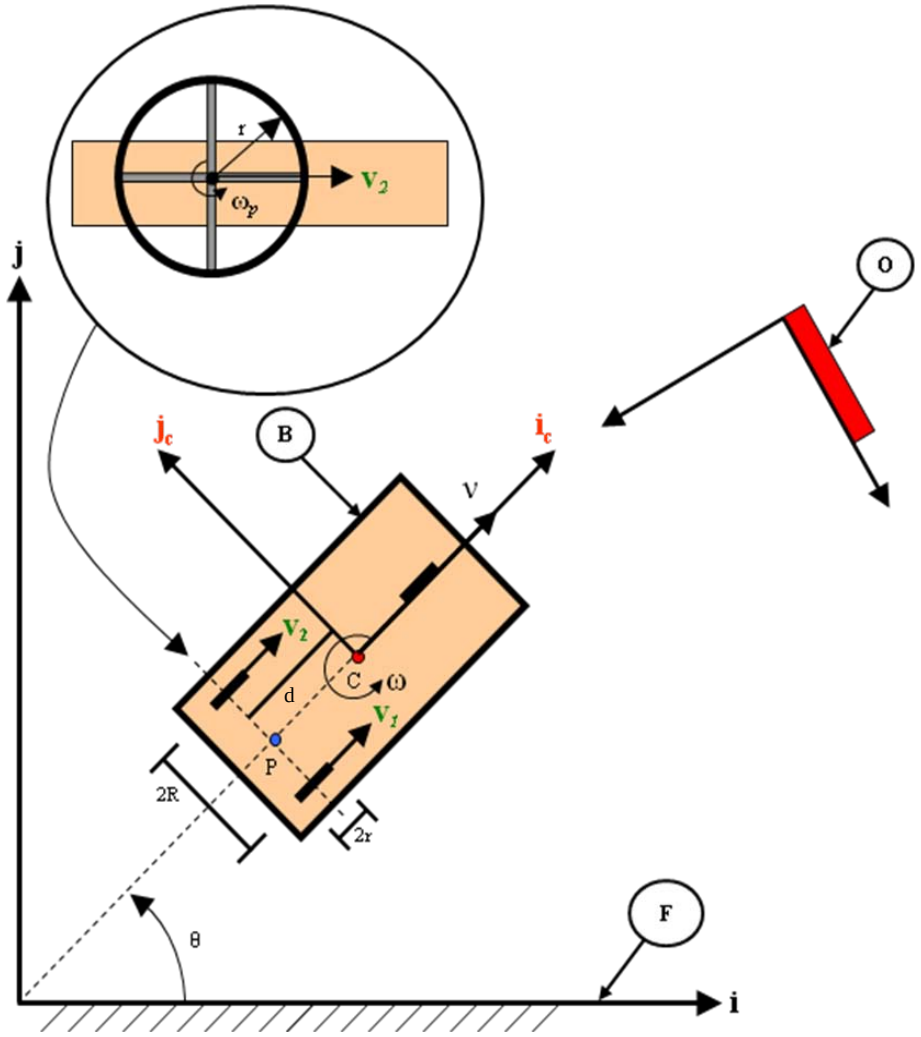


Figure 2.1 A nonholonomic mobile platform. C – Center of mass; v – Velocity at the center of mass; $\mathbf{v_1}$, $\mathbf{v_2}$, – Linear velocities at the center of the right and left wheels, respectively; θ – The orientation of the robot with respect to the fixed frame F.

If we use the notation of the generalized coordinates $\mathbf{q} = (x, y, \theta)^T$ to describe the x-y position of a reference point P on the robot and the orientation of the robot in an inertial Cartesian coordinate frame (frame F in Fig. 2.1), the generalized velocity of the robot can be computed as

$$\dot{\mathbf{q}} = \begin{bmatrix} \cos\theta & -d\sin\theta \\ \sin\theta & d\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{2.3}$$

The nonholonomic constraints of the robot which describe the non-slipping condition along the axis of the driving wheels can be written as

$$\mathbf{A}(\mathbf{q})\dot{\mathbf{q}} = 0, \tag{2.4}$$

where

$$\mathbf{A}(\mathbf{q}) = \begin{bmatrix} -\sin\theta \\ \cos\theta \\ -d \end{bmatrix}. \tag{2.5}$$

2.2 Dynamics

The dynamics equation for the mobile robot depicted in Fig. 2.1 can be written as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}_m(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \tau_d = \mathbf{B}(\mathbf{q})\tau - \mathbf{A}^T(\mathbf{q})\lambda, \tag{2.6}$$

where $\mathbf{M}(\mathbf{q}) \in \Re^{3\times3}$ is a symmetric, positive definite inertia matrix, $\mathbf{H}_m(\mathbf{q}, \dot{\mathbf{q}}) \in \Re^{3\times3}$ is the centripetal and coriolis matrix, $\mathbf{F}(\dot{\mathbf{q}}) \in \Re^{3\times1}$ denotes the surface friction, $\mathbf{G}(\mathbf{q}) \in \Re^{3\times1}$ is the gravitational vector, τ_d denotes bounded unknown disturbances which includes unstructured unmodeled dynamics, $\mathbf{B}(\mathbf{q}) \in \Re^{3\times2}$ is the input transformation matrix,

$\tau \in \Re^{3\times1}$ is the input vector, and $\lambda \in \Re^{3\times1}$ is the vector of constraint forces. For a mobile robot moving on a horizontal plane, $\mathbf{G}(\mathbf{q}) = 0$.

Considering the nonholonomic constraints of the robot given by Equation (2.5), let $\mathbf{S}(\mathbf{q})$ be a full rank matrix formed by a set of smooth and linearly independent vector fields spanning the null space of $\mathbf{A}(\mathbf{q})$, which can be written as

$$\mathbf{S}^T(\mathbf{q})\mathbf{A}^T(\mathbf{q}) = 0, \tag{2.7}$$

where
$$\mathbf{S}(\mathbf{q}) = \begin{bmatrix} \cos\theta & -d\sin\theta \\ \sin\theta & d\cos\theta \\ 0 & 1 \end{bmatrix}, \tag{2.8}$$

The auxiliary vector time function $\mathbf{v}(t) \in \Re^1$ can be defined for all *t* as

$$\dot{\mathbf{q}} = \mathbf{S}(\mathbf{q})\mathbf{v}(t), \tag{2.9}$$

where
$$\mathbf{v} = \begin{bmatrix} \nu \\ \omega \end{bmatrix} \tag{2.10}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -d\sin\theta \\ \sin\theta & d\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \nu \\ \omega \end{bmatrix}, \tag{2.11}$$

where $|\nu| \le \nu_{\max}$ and $|\omega| \le \omega_{\max}$. $\nu_{\max}$ and $\omega_{\max}$ are the maximum linear and angular velocities of the robot in consideration. Equations (2.8), (2.10), and (2.11) constitute the *steering system* of the vehicle.

By multiplying (2.6) by $\mathbf{S}^{\mathrm{T}}$, we can eliminate the constraint matrix $\mathbf{A}^T(\mathbf{q})\lambda$ noting the relation given by (2.7). After replacing $\dot{\mathbf{q}}$ with the relation given by (2.9), the new dynamics equation becomes

$$\mathbf{S}^T\mathbf{M}\mathbf{S}\dot{\mathbf{v}} + \mathbf{S}^T(\mathbf{M}\dot{\mathbf{S}} + \mathbf{H}_m\mathbf{S})\mathbf{v} + \mathbf{S}^T\mathbf{F}\mathbf{S} + \bar{\tau}_d = \mathbf{S}^T\mathbf{B}\tau. \tag{2.12}$$

We can rewrite this as

$$\overline{\mathbf{M}}(\mathbf{q})\dot{\mathbf{v}} + \overline{\mathbf{H}}_m(\mathbf{q},\dot{\mathbf{q}})\mathbf{v} + \overline{\mathbf{F}}(\mathbf{v}) + \bar{\tau}_d = \overline{\mathbf{B}}\tau, \tag{2.13}$$

$$\bar{\tau} = \overline{\mathbf{B}}\tau, \tag{2.14}$$

where $\overline{\mathbf{M}}(\mathbf{q}) = \mathbf{S}^T\mathbf{M}\mathbf{S} \in \Re^{2\times 2}$ is a symmetric, positive definite inertia matrix, $\overline{\mathbf{H}}_m(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{S}^T(\mathbf{M}\dot{\mathbf{S}} + \mathbf{H}_m\mathbf{S}) \in \Re^{2\times 2}$ is the centripetal and coriolis matrix, $\overline{\mathbf{F}}(\mathbf{v}) = \mathbf{S}^T\mathbf{F}\mathbf{S} \in \Re^{2\times 1}$ is the surface friction, $\bar{\tau}_d$ denotes bounded unknown disturbances, $\bar{\tau} \in \Re^{2\times 1}$ is the input vector, and $\overline{\mathbf{B}} = \mathbf{S}^T\mathbf{B} \in \Re^{2\times 2}$ is a constant nonsingular matrix dependent on the distance between the driving wheels **R** and the radius of the wheel **r** (refer to Fig. 2.1). Equation (2.13) describes the behavior of the nonholonomic system in a set of local coordinates. By inserting (2.9) into the dynamics equation, it is clear that the Jacobian Matrix, **S**(**q**), transforms the mobile base coordinates **v** into Cartesian coordinates $\dot{\mathbf{q}}$.

2.3 Trajectory Tracking Control

The described steering system changes the desired **v** into a torque control, τ, for the actual mobile platform. If **u** is an auxiliary input, we can apply the nonlinear feedback to obtain

$$\tau(t) = f_\tau(\mathbf{q},\dot{\mathbf{q}},\mathbf{v},\mathbf{u}) = \overline{\mathbf{B}}^{-1}(\mathbf{q})\left[\overline{\mathbf{M}}(\mathbf{q})\mathbf{u} + \overline{\mathbf{H}}_m(\mathbf{q},\dot{\mathbf{q}})\mathbf{v} + \overline{\mathbf{F}}(\mathbf{v})\right], \tag{2.15}$$

where one can convert the dynamic control problem into the kinematic control problem.

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{S}(\mathbf{q})\mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{u}\end{aligned}. \tag{2.16}$$

The above equation represents a state-space description of the nonholonomic mobile robot. It also constitutes the basic framework for defining its nonlinear control properties. It is assumed in (2.15) that all the dynamical quantities of the robot $\left(\overline{\mathbf{M}}(\mathbf{q}), \overline{\mathbf{F}}(\mathbf{v}), \overline{\mathbf{H}}_m(\mathbf{q}, \dot{\mathbf{q}})\right)$ are known and $\bar{\tau}_d = 0$. The objective is to select the torque in (2.15) so that (2.16) will exhibit the desired behavior motivating the specific choice of the velocity $\mathbf{v}(t)$. This allows the steering system commands to be converted into torques that take into account all of the actual robot's parameters.

There are three basic problems that the nonholonomic steering $\mathbf{v}(t)$ may be divided into: tracking a reference trajectory, following a path, and point stabilization. Since we will be dealing with reference trajectory tracking, only the first problem will be presented in this paper. For more information on path following and point stabilization, refer to [9].

Tracking a Reference Trajectory

The trajectory tracking problem is described as follows: Let there be a reference robot with position and velocity values

$$\mathbf{q}_r = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}, \mathbf{v}_r = \begin{bmatrix} \nu_r \\ \omega_r \end{bmatrix}$$

$$\begin{aligned} \dot{x}_r &= \nu_r \cos\theta_r \\ \dot{y}_r &= \nu_r \sin\theta_r \\ \dot{\theta}_r &= \omega_r \end{aligned} \quad , \qquad \textbf{(2.17)}$$

where $\nu_r > 0$ for all t. There is a smooth velocity control $\mathbf{v}_c(t) = f_c(\mathbf{e}_p, \mathbf{v}_r, \mathbf{C})$ such that $\lim_{t\to\infty}(\mathbf{q}_r - \mathbf{q}) = 0$ where $\mathbf{e}_p$, $\mathbf{v}_r$, and $\mathbf{C}$ are the tracking error, reference velocity vector, and

the controller gain vector, respectively. The torque input, $\tau(t)$ for (2.15), is then computed such that $\mathbf{v} \to \mathbf{v}_c$ as $t \to \infty$. Assuming that the dynamics of the robot is completely known, (2.15) is used to compute the torque, given $\mathbf{u}(t)$. From here, we present the *integrator backstepping method* [9] to derive a suitable $\mathbf{u}(t)$ and $\tau(t)$ from a specific $\mathbf{v}_c(t)$ that controls the steering system (2.16).

It is assumed that the solution to the steering system tracking problem is available and is denoted by $\mathbf{v}_c(t)$. The tracking error vector is expressed as

$$\mathbf{e}_p = \mathbf{T}_e(\mathbf{q}_r - \mathbf{q}),$$

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}, \tag{2.18}$$

and its derivative is expressed as

$$\dot{\mathbf{e}}_p = \begin{bmatrix} \omega e_2 - \nu + \nu_r \cos e_3 \\ -\omega e_1 + \nu_r \sin e_3 \\ \omega_r - \omega \end{bmatrix}. \tag{2.19}$$

The equations for the auxiliary velocity control input that achieves tracking for (2.16) is given by

$$\mathbf{v}_c = \begin{bmatrix} \nu_r \cos e_3 + c_1 e_1 \\ \omega_r + c_2 \nu_r e_2 + c_3 \nu_r \sin e_3 \end{bmatrix}, \tag{2.20}$$

and whose derivative becomes

$$\dot{\mathbf{v}}_c = \begin{bmatrix} \dot{\nu}_r \cos e_3 \\ \dot{\omega}_r + c_2 \dot{\nu}_r e_2 + c_3 \dot{\nu}_r \sin e_3 \end{bmatrix} + \begin{bmatrix} c_1 & 0 & -\nu_r \sin e_3 \\ 0 & c_2 \nu_r & c_3 \nu_r \cos e_3 \end{bmatrix} \dot{\mathbf{e}}_p. \tag{2.21}$$

The proposed nonlinear feedback acceleration control input is

$$\mathbf{u} = \dot{\mathbf{v}}_c + \mathbf{C}_4(\mathbf{v}_c - \mathbf{v}), \tag{2.22}$$

where $\mathbf{C}_4$ is a positive, definite, diagonal matrix,

$$\mathbf{C}_4 = c_4\mathbf{I}. \tag{2.23}$$

By defining an auxiliary velocity error as

$$\mathbf{e}_c = \mathbf{v} - \mathbf{v}_c,$$
$$\mathbf{e}_c = \begin{bmatrix} e_4 \\ e_5 \end{bmatrix} = \begin{bmatrix} \nu - \nu_c \\ \omega - \omega_c \end{bmatrix} = \begin{bmatrix} \nu - \nu_r \cos e_3 - c_1 e_1 \\ \omega - \omega_r - c_2 \nu_r e_2 - c_3 \nu_r \sin e_3 \end{bmatrix}, \tag{2.24}$$

and by using (2.22), we obtain

$$\dot{\mathbf{e}}_c = -\mathbf{C}_4 \mathbf{e}_c, \tag{2.25}$$

and the velocity vector of the mobile base then satisfies $\mathbf{v} \to \mathbf{v}_c$ as $t \to \infty$**.**

By the assumption $\nu_r > 0$, then $e_2 \to 0$ as $t \to \infty$. Therefore, the equilibrium point $\mathbf{e} = 0$ is uniformly asymptotically stable. For more information, refer to [9].

Simulation Results

We simulate a robot with the distance between the wheels R = 0.5m, the distance between points P and C d = 0m, and the wheel radius r = 0.05m. The mass and the inertia of the robot are 10kg and 5kgm^2, respectively. The initial positions and velocities of the robot are set to zero.

Our first simulation example is to have the robot follow a straight line trajectory that is perpendicular to its initial orientation. The trajectory is given in the form of a reference robot with the following initial conditions

$$\begin{pmatrix} \mathbf{q}_r \\ \mathbf{v}_r \end{pmatrix} = \begin{pmatrix} 1 & m \\ 0 & m \\ 90 & \text{deg} \\ 0.5 & m/s \\ 0 & rad/s \end{pmatrix}$$

with the robot's initial conditions set as

$$\begin{pmatrix} \mathbf{q} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} 2 & m \\ 2 & m \\ 0 & \deg \\ 0 & m/s \\ 0 & rad/s \end{pmatrix}.$$

The simulation results are shown in Fig. 2.2. The mobile robot would begin its motion by moving in reverse to reposition itself to move toward the reference robot. The motion of the robot results in convergence with the reference robot's trajectory.

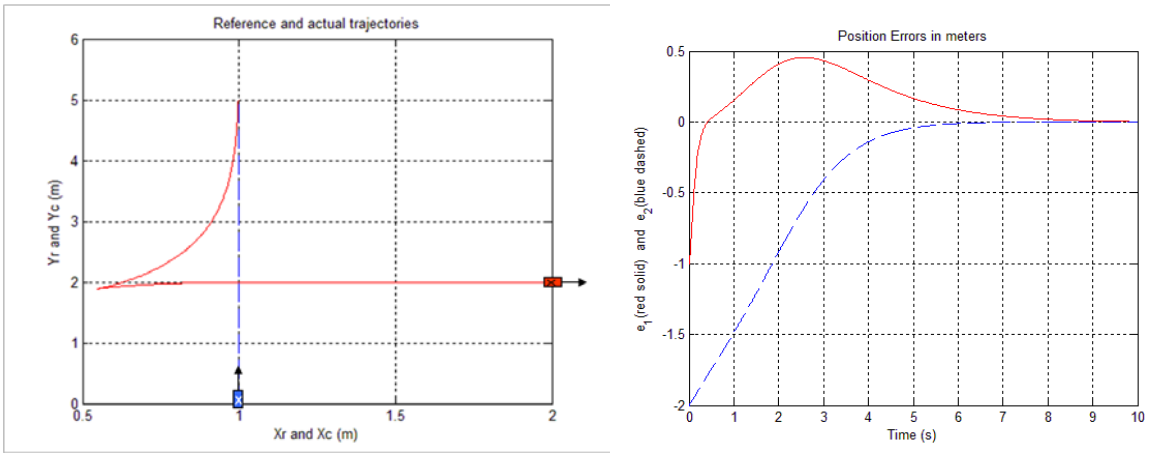
Our second example is to have the robot follow a semicircular trajectory with a radius of 2m over a period of ten seconds. The initial conditions of the reference robot are

$$\begin{pmatrix} \mathbf{q}_r \\ \mathbf{v}_r \end{pmatrix} = \begin{pmatrix} 0 & m \\ 0 & m \\ 90 & \deg \\ \pi/5 & m/s \\ \frac{\pi}{10} & rad/s \end{pmatrix}$$

with the robot's initial conditions set as

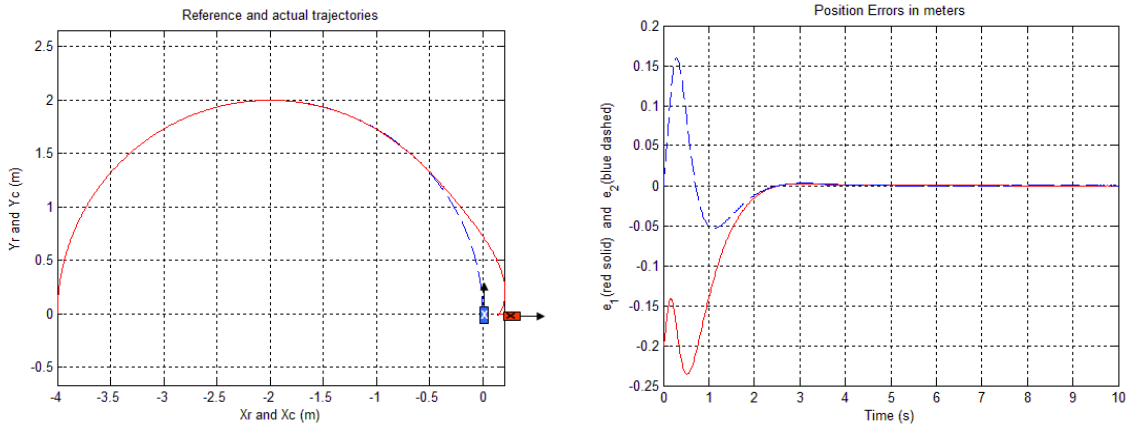
$$\begin{pmatrix} \mathbf{q}_0 \\ \mathbf{v}_0 \end{pmatrix} = \begin{pmatrix} 0.2 & m \\ 0 & m \\ 0 & \deg \\ 0 & m/s \\ 0 & rad/s \end{pmatrix}.$$

The simulation results are shown in Fig. 2.3. The mobile robot would begin its motion by moving in reverse to reposition itself to move toward the reference robot. Once the robot completes the reverse motion, it would begin following the path of the reference trajectory.



(a) **(b)**

Figure 2.2 A differential drive mobile robot following a straight line reference trajectory using the backstepping control. **(a)** Reference robot's path (dashed) and mobile robot's response (solid); **(b)** and error in position of the mobile robot over time in the x- and y-directions.



(a) **(b)**

Figure 2.3 A differential drive mobile robot following a semicircular reference trajectory using the backstepping control. **(a)** Reference robot's path (dashed) and mobile robot's response (solid); **(b)** and error in position of the mobile robot over time in the x- and y-directions

The results from both examples show that as $t \to \infty$, the mobile robot trajectory converges to the reference trajectory, resulting in a decrease in both position and orientation errors between the mobile robot and the specified reference trajectory.

2.4 Discussion

This Chapter provides a stable control algorithm based on the backstepping method that is capable of dealing with reference trajectory tracking problems. Our simulations and experiments show that the kinematic control is able to stabilize a nonholonomic mobile robot about a reference trajectory when the reference trajectory has a constant or varying velocity profile. This servo-control scheme is valid as long as the velocity control inputs are smooth and bounded and the dynamics of the robot is completely known. In practice, the perfect knowledge of the inertial property is not easy to obtain. The neural network based adaptive control approach may be able to learn these parameters online and may resolve this problem.

The kinematics and dynamics equations are linked by (2.15). The steering system, along with the actual and reference robot's position and velocities (i.e., the kinematics of both vehicles), is used in the integrator backstepping method to solve for the auxiliary control velocity, $\mathbf{v}_c$, and the feedback acceleration control input, $\mathbf{u}$, to reduce the position and velocity error between the reference and actual robot. These values are plugged into (2.15) and the wheel torques can be calculated assuming the initial properties are known.

Chapter 3

Vision-Based Sensing

For the vision system of the robot, we consider a widely used robot configuration with one on-board camera. This configuration is similar to the eye-in-hand configuration in a visual servoing system [10]. The difference between the eye-in-hand system and the system used here is that instead of the camera being mounted on the hand of a robot manipulator which is fixed at the base, the camera is attached to a mobile platform that can move freely in its workspace. In order to sense and eventually track an object, camera calibration must be performed to determine the intrinsic parameters of the camera such as the focal length of the optical system, principal point, skew coefficients, and distortions.

In this chapter, we describe the model utilized for camera calibration. This is followed by the implementation details and experiments to validate the accuracy of the intrinsic parameter recognitions. In the experiments, we calculate the transformation matrix of an object from the camera's perspective in two known orientations and compare this matrix to the known transformation matrix.

3.1 Models for Camera Calibration

Camera calibration is a technique in computer vision which is used to determine a camera's internal geometric and optical characteristics (intrinsic parameters) and the 3D

position and orientation of the camera system relative to a certain world coordinate system (extrinsic parameters). In order to identify the intrinsic parameters of the camera, we use a method called Direct Linear Transformation (DLT). This method works by taking several snapshots of the camera's view with an object of a known geometry and location relative to the camera. In each snapshot, the object is moved to a different location and with a different orientation relative to the camera [11]. These snapshots are then used to determine the intrinsic parameters of the camera. In general, the intrinsic parameters of a vision system include:

- Focal length (f_c): The focal length (in pixels) is stored as a 2x1 vector.
- Principal point (p_c): The principal point coordinates are stored as a 2x1 vector.
- Skew Coefficient (α_c): The skew coefficient defining the angle between the x and y pixel axes is stored as a scalar value.
- Distortions (k_c): The image distortion coefficients (radial and tangential) are stored as a 5x1 vector.

3.1.1 DLT Model

For the DLT model, we assume that there is no image distortion (i.e. $k_c = 0$). We start with an initial point, **P**, where

$$\mathbf{P}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}, \qquad \mathbf{P}_o = \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} \tag{3.1}$$

and $$\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} = \mathbf{R}_c\mathbf{P}_c + \mathbf{d} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} \mathrm{X_c} \\ \mathrm{Y_c} \\ \mathrm{Z_c} \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}, \quad \textbf{(3.2)}$$

where $\mathbf{P}_o$ is a vector containing the 3D coordinates of point **P** in the object reference frame, $\mathbf{P}_c$ is a vector containing the 3D coordinates of point **P** in the camera reference frame, **d** is the translation vector from the camera frame of reference to the object frame of reference, and $\mathbf{R}_c$ is the rotation matrix from the camera reference frame to the object reference frame (refer to Fig. 3.1 and 3.2).

Since the camera reference frame has the same orientation as the body-fixed reference frame of the robot which the camera is attached to, they differ by only a translation. They are related by

$$\mathbf{P}_B = \mathbf{P}_c + \mathbf{d}_c, \quad \textbf{(3.3)}$$

where $\mathbf{d}_c$ is the translation vector in the camera reference frame.

We can define N_n to be the normalized (pinhole) image projection

$$\mathbf{N}_n = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{f_c}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix}, \quad \textbf{(3.4)}$$

where u_i and v_i are the $\mathrm{X_c}$ and $\mathrm{Y_c}$ locations of the point $\mathbf{P}_c$ projected on a 2-dimensional image plane, respectively (see Fig. 3.1). This frame of reference is what is seen by the camera. The given values for this model are $\mathbf{P}_c$ and $\mathbf{P}_o$. After obtaining the corner locations of at least five points on the object for each orientation, enough information has

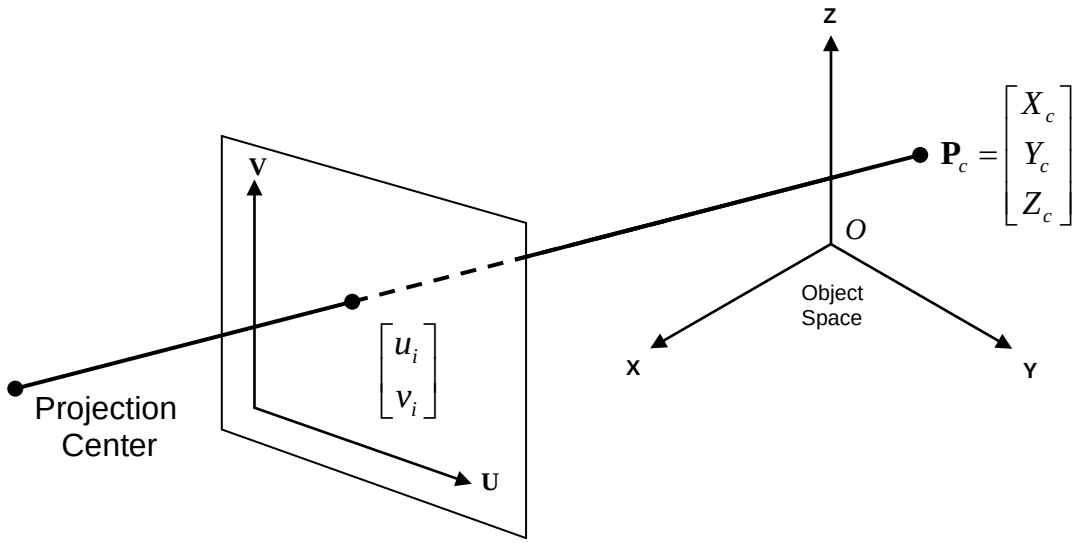
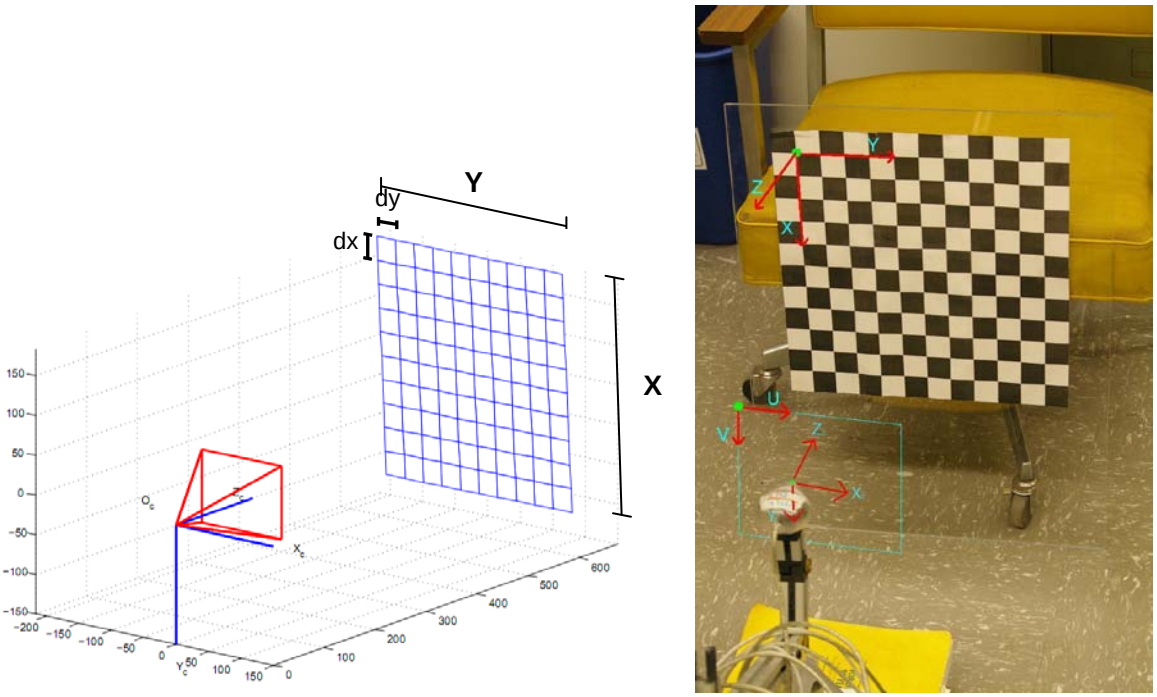


Figure 3.1 Projection of the point P_c on the image plane



(a) **(b)**

Figure 3.2 (a) Setup for the calculation of intrinsic parameters. X, Y, dx, dy are predefined; **(b)** and camera reference frame (X_c, Y_c, Z_c), Object reference frame (X, Y, Z), and Image plane reference frame (U, V). Note that the object reference frame is labeled without the "o" subscript to remove confusion in this figure between X_o, Y_o, Z_o and X_c, Y_c, Z_c.

been collected to determine the intrinsic parameters. From here, we can easily solve for f_c.

Although the DLT model is computationally efficient, it is not always accurate enough to use. This is due to not taking lens distortions into account.

3.1.2 DLT Model with Distortion Compensation

Although the lens distortion model takes image warping into account, it is a time consuming method and the possibility of getting trapped in a local error minimum arises [12]. To remedy these issues, we use a combination of the DLT and distortion models. This applies the distortion-free camera model to the input data and uses the results to apply a nonlinear search of each local corner location, resulting in a method that takes distortion into account and removes any local minimum traps.

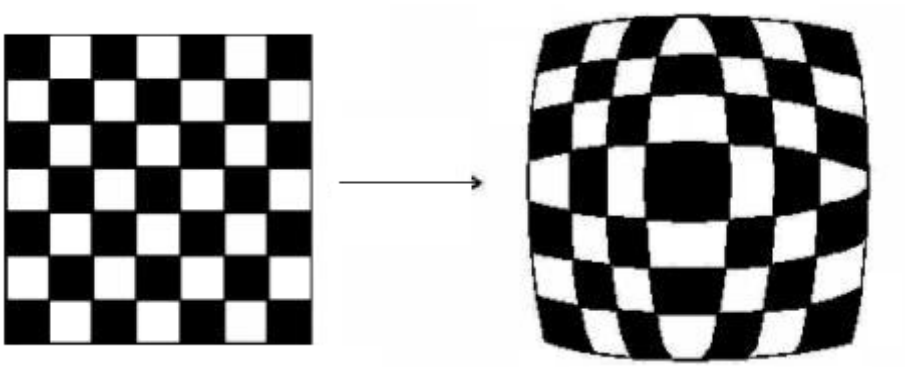
The normalized point with lens distortion taken into account is written as

$$\mathbf{N}_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix} = \left(1 + k_{c1}t^2 + k_{c2}t^4 + k_{c5}t^6\right)\mathbf{N}_n + \mathbf{dx}, \tag{3.5}$$

where $t^2 = u_i^2 + v_i^2$ and

$$\mathbf{dx} = \begin{bmatrix} 2k_{c3}u_i v_i + k_{c4}(t^2 + 2u_i^2) \\ k_{c3}(t^2 + 2v_i^2) + 2k_{c4}u_i v_i \end{bmatrix}, \tag{3.6}$$

where, for the tangential distortion vector **dx**, k_{c1}, k_{c2}, and k_{c5} are radial distortions and k_{c3} and k_{c4} are tangential distortions. From here, we can relate the pixel coordinates to the



Actual image Distorted image

Figure 3.3 Comparison of Actual image to a Distorted image as seen by a camera with a distorted lens

projection of point $\mathbf{P_c}$ on the image plane

$$\begin{cases} x_p = f_{c1}(x_d + \alpha_c y_d) + p_{c1} \\ y_p = f_{c2} y_d + p_{c2} \end{cases}, \tag{3.7}$$

where x_p and y_p are the final coordinates in pixels of the projection of point **P** on the image plane. Since the skew coefficient determines the relationship between the u-axis and v-axis on the image plan, the coefficient only needs to be accounted for in one of the two terms (refer to Fig. 3.4).

Equation (3.7) can be rewritten as

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \mathbf{L} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}, \tag{3.8}$$

where

$$\mathbf{L} = \begin{bmatrix} f_{c1} & \alpha_c f_{c1} & p_{c1} \\ 0 & f_{c2} & p_{c2} \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

After substituting all of the known variables into (3.7), we obtain

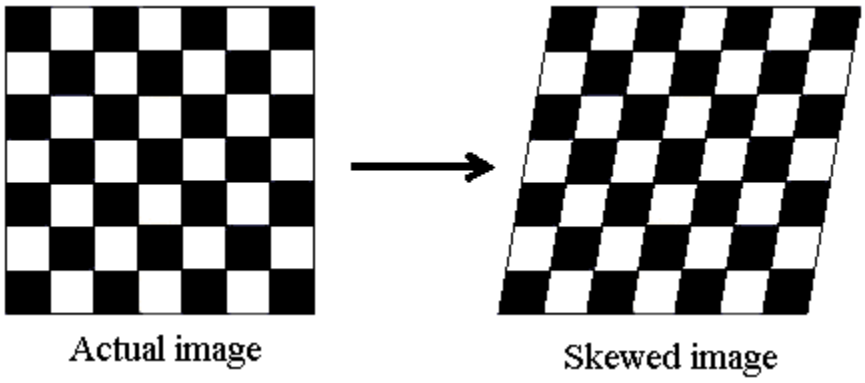


Figure 3.4 Comparison of Actual image and the same image as seen by a camera with a skewed perspective

$$x_p = f_{c1}((1+k_{c1}t^2+k_{c2}t^4+k_{c5}t^6)\frac{f_{c1}}{\mathrm{Z_c}}\mathrm{X}_c + 2k_{c3}\left(\frac{f_{c1}f_{c2}}{\mathrm{Z}_c^{\,2}}\right)\mathrm{X}_c\mathrm{Y}_c + k_{c4}(t^2+2\left(\frac{f_{c1}}{\mathrm{Z}_c}\mathrm{X}_c\right)^2) + \alpha_c(1+k_{c1}t^2+k_{c2}t^4+k_{c5}t^6)\frac{f_{c2}}{\mathrm{Z}_c}\mathrm{Y}_c + k_{c3}(t^2+2\left(\frac{f_{c2}}{\mathrm{Z}_c}\mathrm{Y}_c\right)^2) + 2k_{c4}\left(\frac{f_{c1}f_{c2}}{\mathrm{Z}_i^{\,2}}\right)\mathrm{X}_c\mathrm{Y}_c) + p_{c1} \quad \textbf{(3.10)}$$

$$y_p = f_{c2}(1+k_{c1}t^2+k_{c2}t^4+k_{c5}t^6)\frac{f_{c2}}{\mathrm{Z}_c}\mathrm{Y}_c + k_{c3}(t^2+2\left(\frac{f_{c2}\mathrm{Y}_c}{\mathrm{Z}_c}\right)^2) + 2k_{c4}\left(\frac{f_{c1}f_{c2}}{\mathrm{Z}_c^{\,2}}\right)\mathrm{X}_c\mathrm{Y}_c + p_{c2}.$$

These two equations relate to the given coordinates in pixels to the given coordinates in the camera reference frame. Since there are a total of two equations and ten unknowns (2 for f_c, 2 for p_c, 1 for α_c, and 5 for k_c), we need a minimum of five unique points in order to solve for the intrinsic parameters. The more points used, the greater the accuracy of the intrinsic parameters. For more information, refer to [13].

3.1.3 Pose Estimation Using DLT Model

After obtaining the intrinsic parameters, we can calculate the extrinsic parameters for any orientation of any object of known dimensions. For every image of the object that we

wish to obtain the transformation matrix, we already have the values for x_p, y_p, f_c, p_c, α_c, and k_c. In the implementation, we specify an origin of the object frame and the pose of the object is estimated by the image analysis. This ensures that there is a unique homogenous transformation matrix corresponding to any object location.

To solve for the transformation matrix, we use (3.2). From (3.10), we can obtain the values of X_c, Y_c, and Z_c. Rewriting (3.2), we obtain

$$\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}, \tag{3.11}$$

where **T** is a 4x4 transformation matrix containing the extrinsic parameters

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \tag{3.12}$$

From (3.11), we have three equations and six unknowns. For every additional point chosen from the same image, we obtain three more equations, but the number of unknowns remains unchanged. Since we are obtaining the transformation matrix from the camera reference frame to the object reference frame, any additional points do not change the transformation matrix, but the values of $\mathbf{P_c}$ and $\mathbf{P_o}$ change. Once the second point is known in the camera reference frame, the extrinsic parameters can be solved for without any difficulties.

3.2 Implementation and Results for the Vision System Calibration

For implementation of the DLT model with distortion and calculation of the extrinsic parameters, an off-the-shelf webcam is utilized for image capturing as the ‘eye’ of the robot.

3.2.1 Intrinsic Parameters

In order to obtain the intrinsic parameters of the webcam, a simple black and white checkered board is used for calibration. The board size is 12x12 squares, with each square exactly 30 millimeters in length per side.

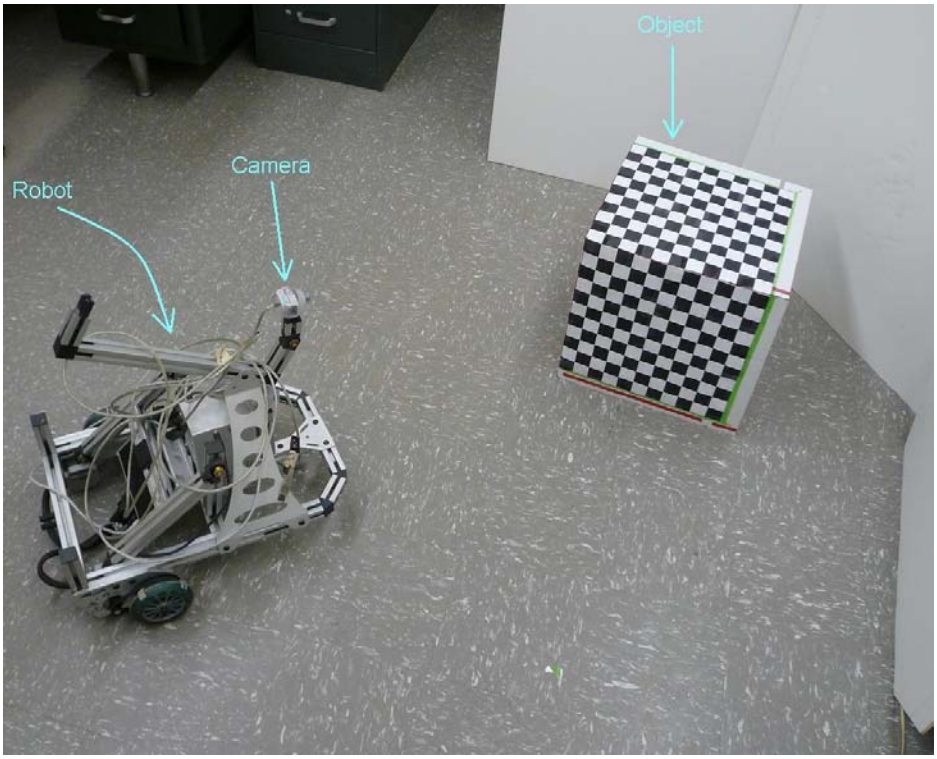
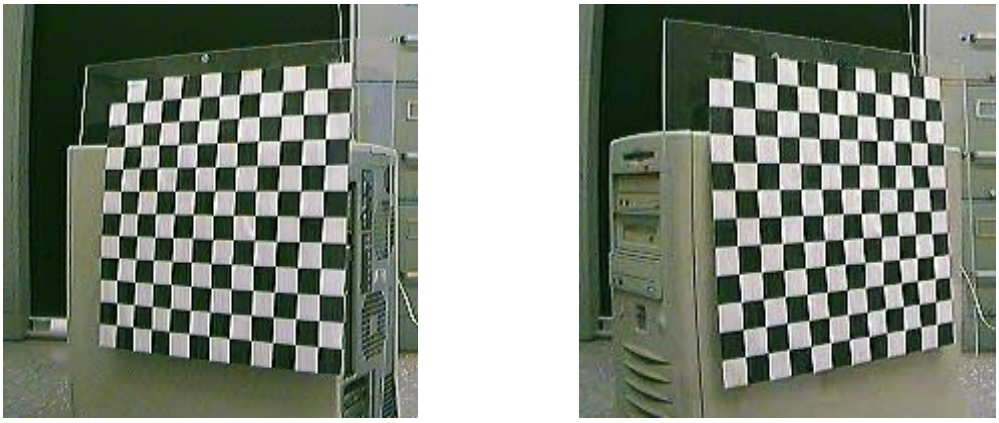


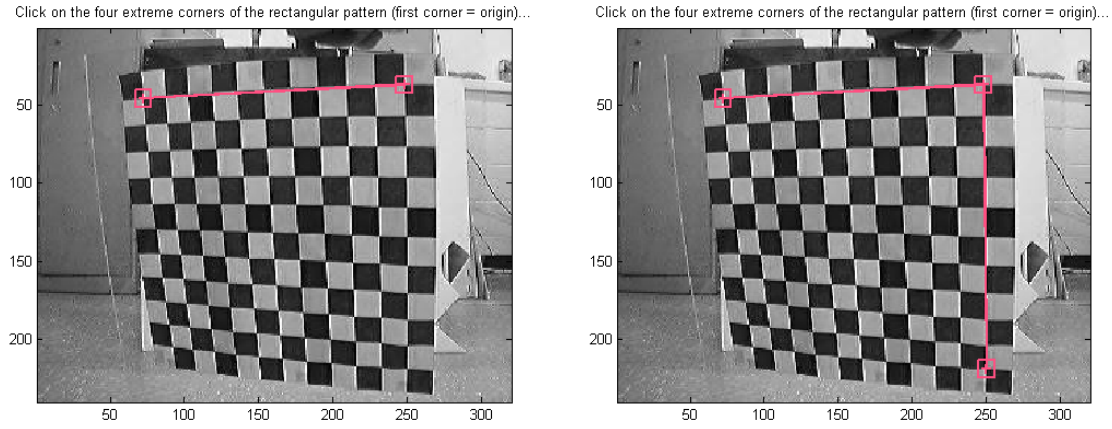
Figure 3.5 General setup for camera calibration.

Our camera calibration program is built on top of the Matlab toolbox described in [14]. In our experiments, 30 images of the checkered board are taken at various locations and orientations from the camera frame. Typical images from the camera are shown in Fig. 3.6. After obtaining at least the minimum required number of images, the four corners of each image is selected as object corner locations as seen in Fig. 3.7. Using the pixel locations of these four corners for all of the images, the intrinsic parameters are calculated by applying the DLT model with distortion as described by Equations (3.5)-(3.10). Fig. 3.8(a) shows the estimated locations prior to using the DLT method (i.e., evenly spaced points). The squares surrounding each corner are the area in which the DLT with distortion method is applied. Fig. 3.8(b) shows the same image after DLT with distortion applied. Fig. 3.9 shows the projected orientation and location of the checkered board for the first 15 images in the order that each image was taken.



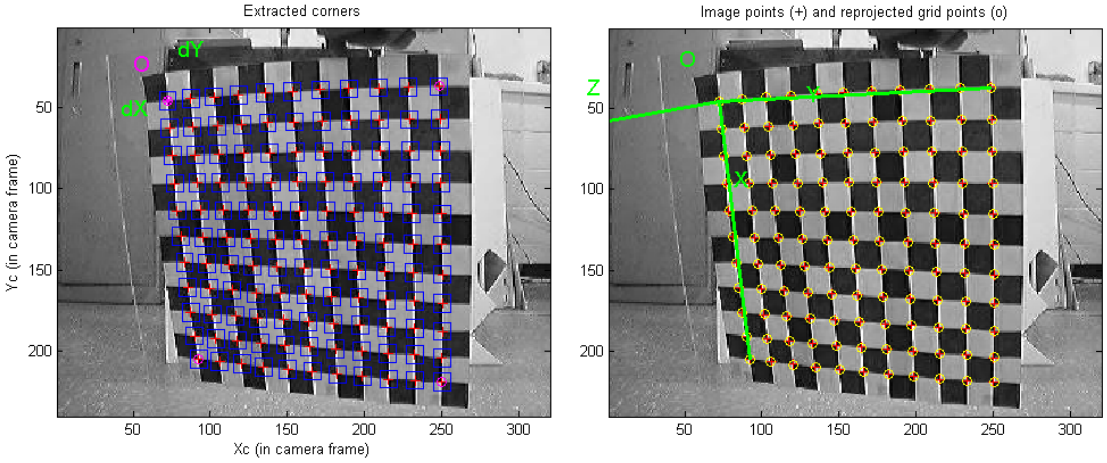
(a) **(b)**

Figure 3.6 Typical images of the calibration grid at different locations and orientations used for camera calibration.



(a) **(b)**

Figure 3.7 Corner capture using the mouse for camera calibration.



(a) **(b)**

Figure 3.8 Checkered board intersect estimations **(a)** Estimated location of each edge; **(b)** and final estimation with distortion taken into account.

3.2.2 Extrinsic Parameters

After the intrinsic parameters are determined, the extrinsic parameters of the object at any location and orientation can be calculated from the camera image. Fig. 3.10 shows the camera image of two calibration grids mounted on two perpendicular surfaces. Since the angle between these two grids is known, we can use this set up to validate our calibration and object identification programs. We will compare the known transformation matrix for

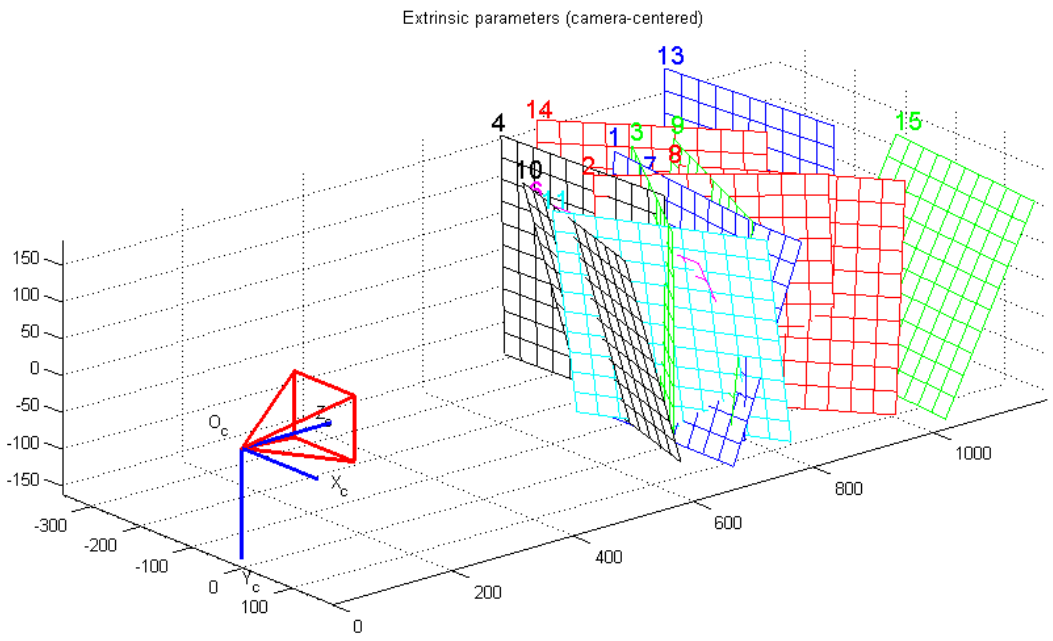
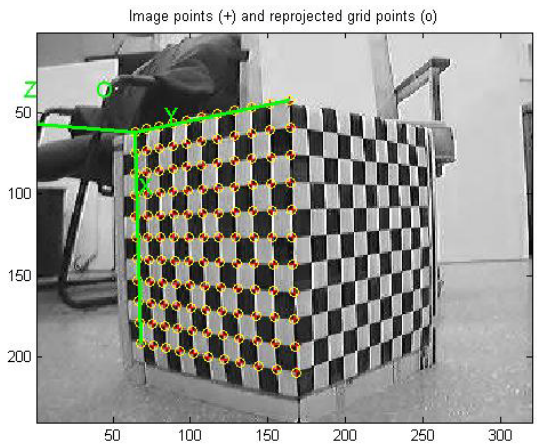


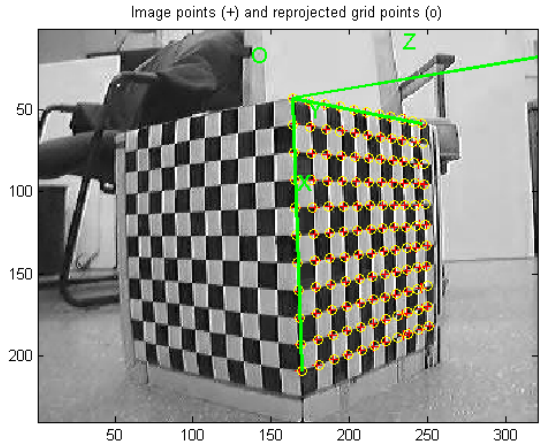
Figure 3.9 Projected locations of each object orientation

the two checkered boards with the two transformation matrices obtained from the checkered board to the camera. Fig. 3.11 shows the results from the extrinsic parameter calculations. The results from the camera image in Fig. 3.10(b) and (c) are indexed as pose 27 and pose 28, respectively. For these two poses, the transformation matrix from the camera to each image is calculated. Since we already know these two objects lie on a cube and the length of each side is known, the transformation from object in Fig. 3.10(c) to the object in Fig. 3.10 (b) (i.e. from pose 28 to pose 27 shown in Fig. 3.11) should ideally be a translation of 300mm along the object's negative z-axis and a rotation of 90 deg. counterclockwise around the object's x-axis

$$ {}^{28}\mathbf{T}_{27} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -300 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad \textbf{(3.13)} $$

(a)



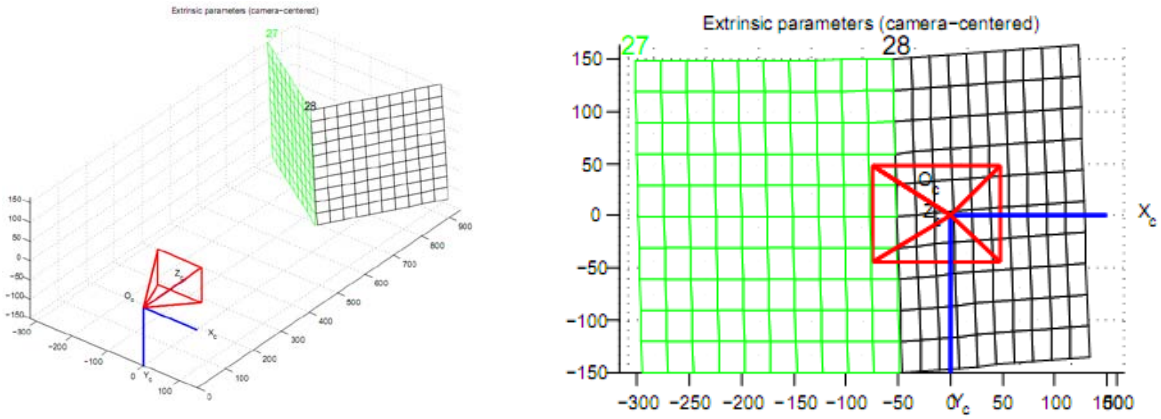


(b) **(c)**

Figure 3.10 (a) Original image used for verification of extrinsic parameter calculations; **(b, c)** Two orientations of the checkered box used to determine the accuracy of extrinsic parameter calculations.

The results from our calibration and identification program are expressed in terms of the transformation matrices from the camera frame to the object frame for each Pose

$$ {}^{C}\mathbf{T}_{27} = \begin{bmatrix} -0.1080 & -0.8016 & -0.5880 & -283.5718 \\ 0.9928 & 0.1174 & -0.0224 & -200.2493 \\ 0.0510 & -0.5862 & -0.8086 & 734.4758 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$



(a) **(b)**

Figure 3.11 Two orientations of the calibration board **(a)** a 3rd person view; **(b)** and on-board camera view.

$$ {}^{C}\mathbf{T}_{28} = \begin{bmatrix} -0.1029 & 0.5997 & 0.7936 & -42.0599 \\ 0.9933 & 0.0198 & 0.1139 & -164.9087 \\ 0.0526 & 0.8000 & -0.5977 & 559.9443 \\ 0 & 0 & 0 & 1 \end{bmatrix}. $$

The resulting transformation from reference frame 28 to reference frame 27 can be calculated as

$$ {}^{28}\mathbf{T}_{27} = \left[{}^{28}\mathbf{T}_{C}\right]\left[{}^{C}\mathbf{T}_{27}\right] = \left[{}^{C}\mathbf{T}_{28}\right]^{-1}\left[{}^{C}\mathbf{T}_{27}\right] = \begin{bmatrix} 1.0000 & 0.0032 & -0.0043 & -1.0717 \\ -0.0043 & 0.0141 & -0.9999 & -5.9172 \\ -0.0032 & 0.9999 & 0.0142 & -300.0034 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(3.14)} $$

Comparing (3.14) to (3.13), we see that the calculated transformation matrices for the two different poses in this example is accurate to within an error of less than 1%.

3.3 Discussion

The method presented in this chapter is used to calibrate the on-board camera of the robot and identify the extrinsic parameters of an object with known geometry. No major issues were encountered with camera calibration. For extrinsic parameter calculations however, issues were encountered for the object detection. These issues are described in section 5.4.

Chapter 4

Experimental Setup

The test bed used in this project is the ER1 Personal Robot System developed by Evolution Robotics™. The ER1 is a differential driven three-wheeled robot with a passive steering wheel. It is purchased as a kit that can be assembled and customized into different configurations.

In this chapter, we present information on the robot design and software interface provided with the ER1. Extensive experiments have been performed to study the capabilities and limitations of the ER1's sensing and motion control systems.

4.1 ER1 Robot

The ER1 robot kit consists of aluminum rods and connectors for the frame, a power module, a Robot Control Module (RCM) for sending commands to and receiving information from the robot, mounting plates (for the power module and RCM), a webcam, and three wheels (one 360-degree rotating caster wheel and two nonholonomic scooter wheels with one stepper motor pre-installed on each). A Dell™ *Inspiron 8200* (Intel Pentium 4 processor 1.8GHz with 512 MB RAM) with Windows XP installed is used as the on-board controller for the robot.

The robot kit provides several suggested assemblies for a variety of applications but can be redesigned into any desired shape and size. This allows the user to customize the robot for better manipulation in unique environments and in multiple surroundings with easier access to the battery, RCM, and/or the on-board computer. Optional add-ons for the robot include a gripper arm for grabbing and transporting objects, 9 infra-red sensors for additional detection options, and a second camera for obstacle avoidance. A customized ER1 robot used in our experiments is shown in Fig. 4.1.

The ER1 comes with software which provides a way to explore the capabilities of the robot (refer to Fig. 4.2). This software is called the Robot Control Center (RCC). The RCC contains a GUI that allows the user to set up if-then commands with a minimal amount of input. Also, because the robot is compatible with many different computer languages (Java, C++, etc.), it is very flexible when dealing with other methods to control the robot.

The RCC software allows the user to make simple motions with the ER1. These

Figure 4.1 Customized ER1 robot

motions can be triggered by a sound, a timer, or even an email. One of these triggers is the use of the webcam. With the webcam, a user can capture an image of an object and store this image in the RCC's memory. By recognizing the stored image in the RCC's memory, the robot can be made to respond to this image in a desired manner.

Another feature that the RCC has is the Application Programmer's Interface (API). This feature allows the user to control the robot directly, rather than by using the GUI. With this feature, the user can input commands to the robot using a command line interface rather than in response to stimuli. However, even with these features, there were still issues with the API that needed to be addressed before achieving usability (see section 4.3 for more information on API issues).

4.2 Vision and Motion Capabilities

As mentioned earlier, the ER1 comes equipped with a webcam for image capturing. The RCC has the capabilities of image capturing, video recording, object recognition, object and color tracking, and obstacle avoidance. For the obstacle avoidance function, a second webcam is required to be aimed toward the floor directly in front of the robot. Several tests were performed to determine the limits of the obstacle avoidance function.

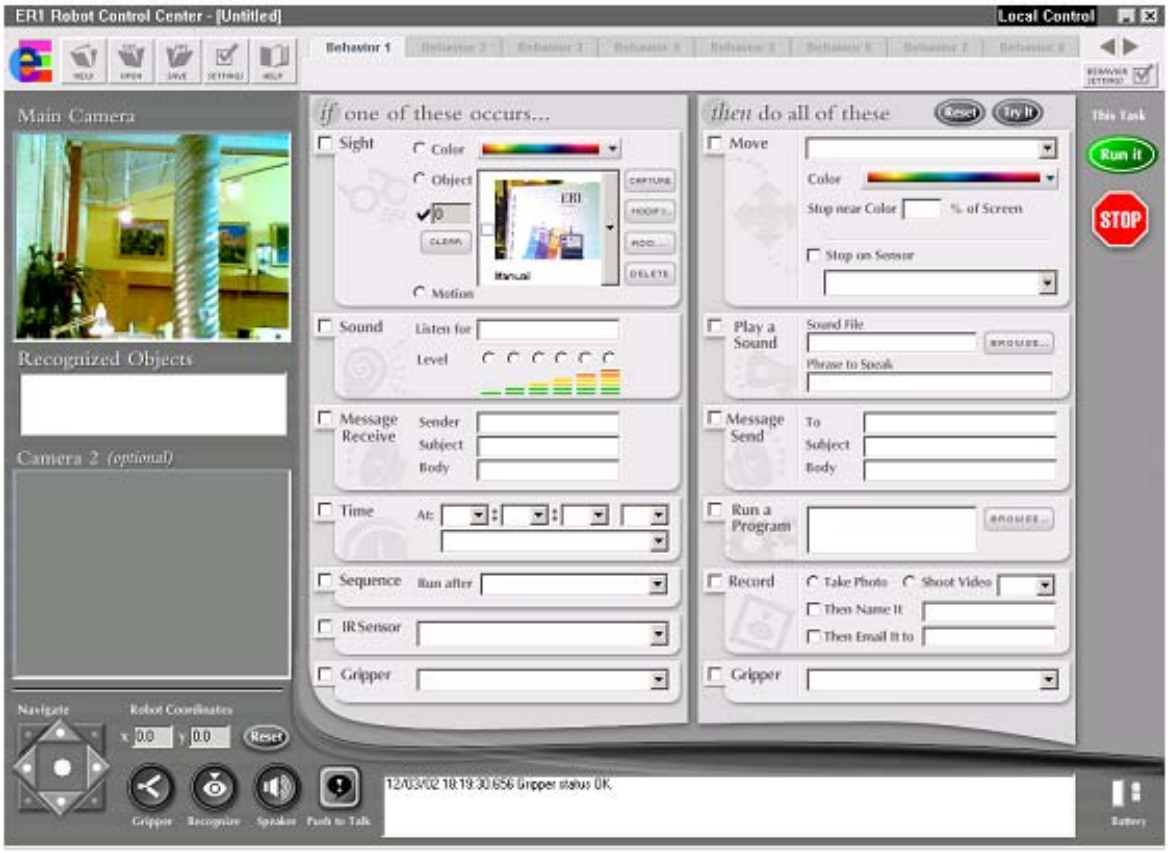


Figure 4.2 Graphical User Interface of the RCC

4.2.1 Obstacle Avoidance

The RCC's obstacle avoidance function can identify obstacles based on either differences in light intensity or differences in color (RGB values). After selecting one of the two choices to use, the tolerance for the choice made can be manipulated. The tolerance range was from 0 to 100, where 0 was as sensitive as possible, and 100 was as tolerant as possible.

Two tests were implemented to determine the robot's obstacle avoidance abilities. In the first test, the robot was given a set distance to move in a straight-line path in which three obstacles were placed. The first two obstacles were chosen to be colors that were

unrelated to the color of the ground. The third obstacle was selected to blend in with the background (refer to Fig. 4.3).

The second test for obstacle avoidance involved giving the robot a nonlinear location to reach. The robot would start in one corner of a rectangular hallway and was commanded to move to the opposite corner of the hallway. In order to do so, the robot would have to avoid the walls of the hallway as the motion was being performed (refer to Fig. 4.4).

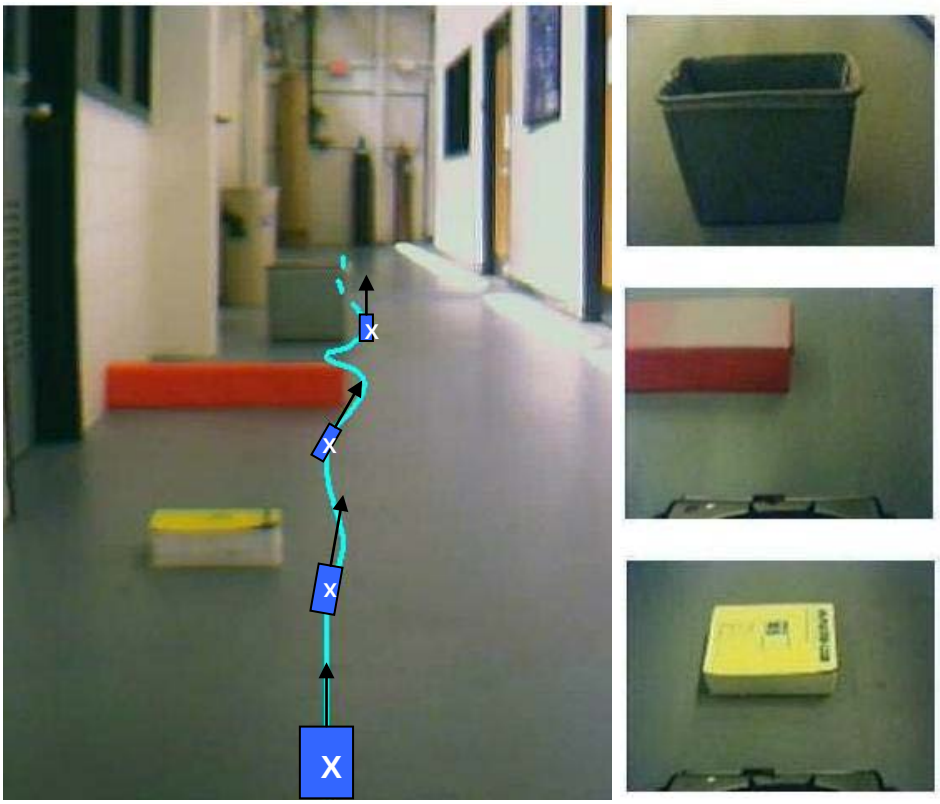
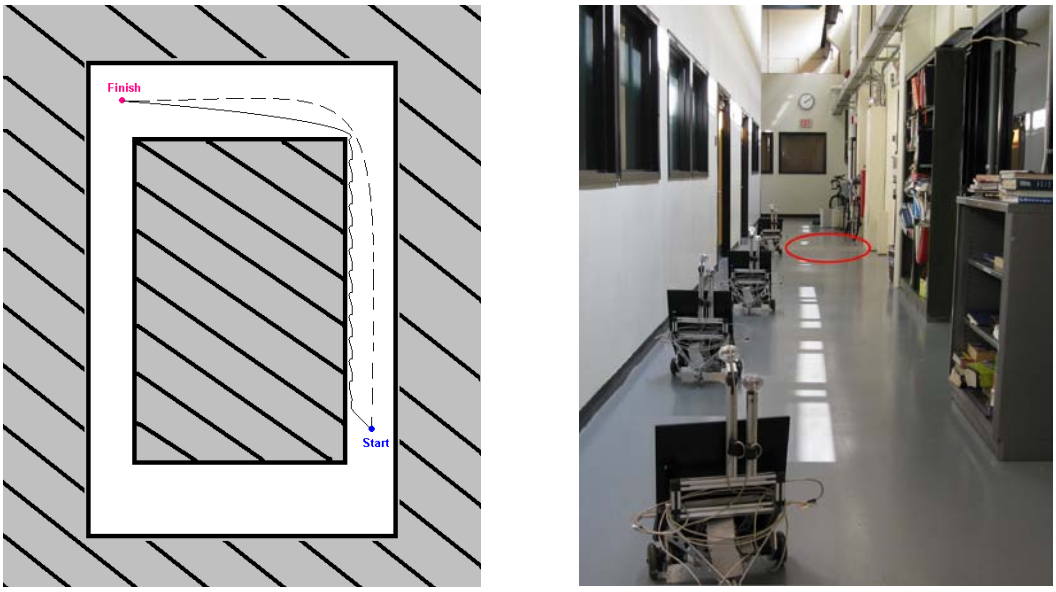


Figure 4.3 Test setup for the first test taken by the robot's camera and a sample path of the robot. The robot was given a straight-line distance to travel in a clustered environment with various obstacles.

Obstacle Avoidance Results The results for the first test showed that both intensity and color difference gave the best results with a tolerance level set between 35 and 50. If the tolerance was set lower than 35, the robot would perform one of two actions: the robot would either detect the floor as an obstacle and keep rotating in place, or its motion

would be extremely jerky and would take a long period of time to complete its task. If the tolerance was set higher than 50, the robot's motion would either be successful, or the robot would either nick or drag the obstacle along the way. Overall, light intensity difference detection gave slightly better results than color difference detection. The color difference detection tests had a larger number of trials that either nicked or dragged the obstacles.

For the second test, regardless of the tolerance level, the robot was unable to complete the required motion. This was due to the ground itself having a section with a grating (see Fig. 4.4(b) and Fig. 4.5). The robot detected the grating as an obstacle and was unable to cross the grating.



(a) **(b)**

Figure 4.4 Second obstacle avoidance test. **(a)** An overhead view of the ideal (dashed line) and predicted (solid line) robot path; **(b)** View of path from starting position. For the second test, the robot was required to navigate the hallway with no prior knowledge other than the location of the endpoint relative to the start position. During this test, the grating (circled) was detected as an obstacle.

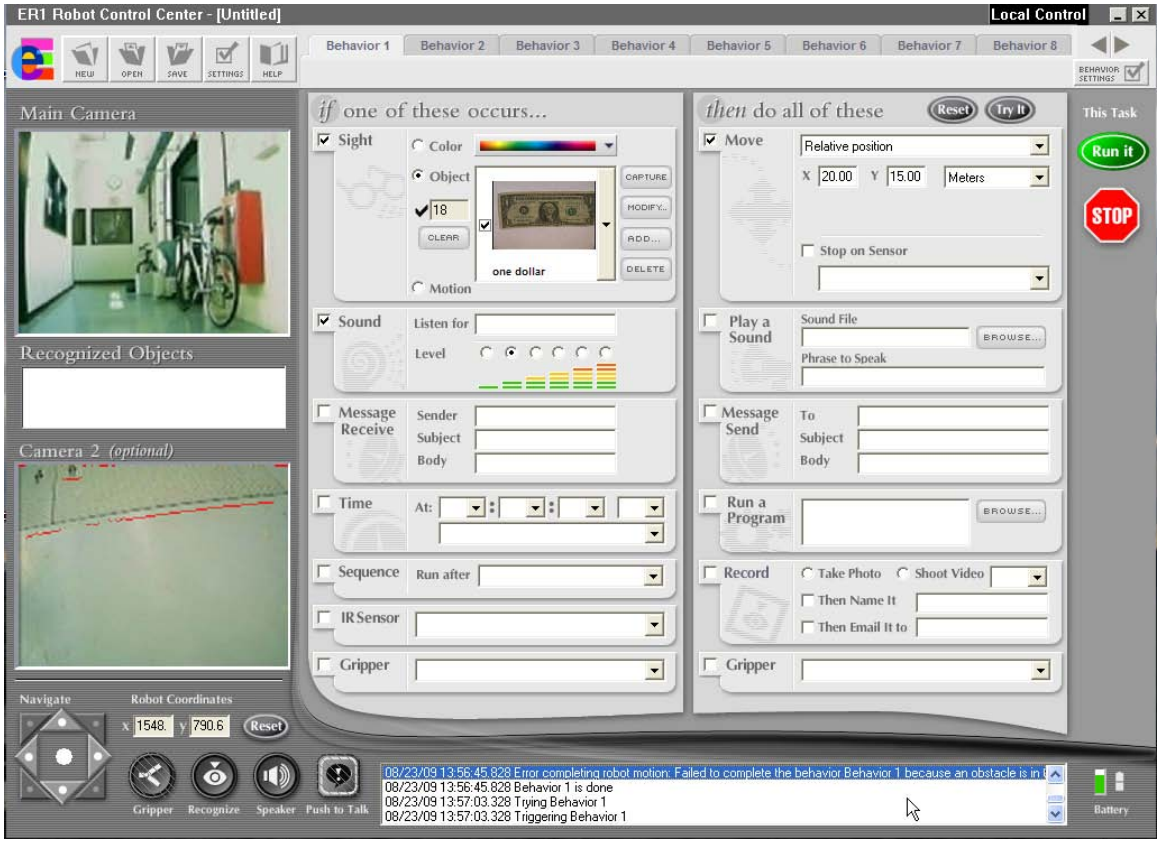


Figure 4.5 View of the grating which acted as an obstacle for the second obstacle avoidance test (marked as "Camera 2"). Note the error message highlighted in blue above the pointer.

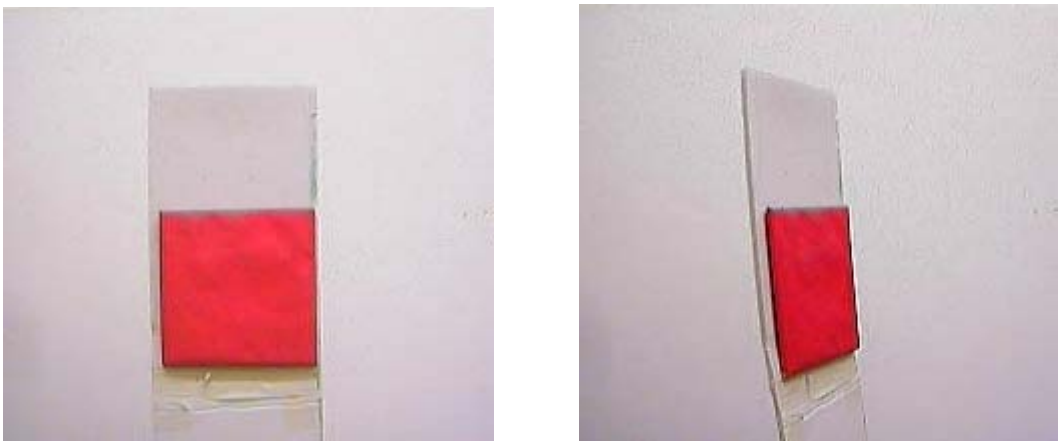
4.2.2 Motion Control System

The ER1 has the ability to be controlled via teleoperation or move autonomously based on input from the camera. By using the recognition software, the robot can be made to move or turn toward or away from an object or color. The ER1 control software imposes limits on the robot with a maximum linear velocity of 50cm/s, and turn at a maximum of 90 deg/s. Although the hardware structure is capable of performing its motions at greater speeds, the limits imposed by the control interface are more than enough to work with for

our purpose. We tested the final position accuracy and some of the discussions are presented in Chapter 5.

4.3 Discussion

ER1 Vision Issues As useful and easy to use as the vision functions were, issues still arose that were unavoidable by using the RCC. One such issue involved motion using color detection. The RCC would determine the motion to or from a user-specified color based on the percentage of the screen taken up by said color. If the object is not seen to be a uniform color by the robot or if the object's orientation reduces the robot's perception of the actual size of the object (refer to Fig. 4.6), the robot would determine the desired final location relative to the object incorrectly. This method of detection was unusable due to the robot's lack of ability to detect the object orientation.



(a) **(b)**

Figure 4.6 (a) Straight; **(b)** and angled object. The object location is unchanged, but due to the orientation, the screen percentage of the object in each image would result in different robot reactions.

Another issue that occurred was image skipping during motion involving a live video feed. During object or color searches, the video feed would, on occasion, freeze for up to several seconds. This would cause the robot to use the latest (and inaccurate)image as the current view. If this happened while the robot was either heading towards a wall or as the target came into view, the robot would miss the target altogether and disaster tended to ensue. Once the video feed would be restored, the robot would continue as though the video had not frozen.

Image Capturing Using Matlab In order to remedy the problems with the RCC, Matlab was used to obtain and analyze image data. With Matlab, frame skipping was greatly reduced and color recognition using camera detection allowed for more accurate distance measurements then with screen percentages.

ER1 Motion Issues The motion for the ER1 was easily manipulated and did not cause any major problems with motion to set location or with tracking of an object or color. One minor issue with the robot motion occurred when specifying x-y coordinates to move towards. If the robot were given a linear distance to move, the ER1 would always finish its motion short of where it had intended to reach. This error was constant and would not increase beyond a certain value. The error value was a relatively small amount compared to the actual and commanded distances traveled by the robot. Section 5.3.2 contains discussions on this error.

Issues With the API There have been several issues while working with the API. The first issue is with the position readout from the robot. Although the x-y position is easily read from the encoders, the angle of the robot is always the value from the robot's frame of reference (i.e. always zero). The second issue is that, although the user guide for the robot provided the commands that can be used for the API, a small number of commands were not mentioned in the user guide and no notification of the availability of these commands was mentioned. This includes one of the most important of the movement commands which involves nonlinear motion to specified x-y coordinates allowing the motion to be completed in a single, fluid action. Without this command, the robot needs to move in straight paths, only turning in place, causing a seemingly unnatural rigid motion.

Another issue was related to the default x-y directions for the robot's motion. As shown in Fig. 3.2, the object's frame of reference and the camera's frame of reference have different orientations and must be taken into account when calculating the motion of the robot. What makes the motion even more difficult is the RCC's choice of the robot's x- and y-directions for motion. The robot's x-direction is in the camera's z-direction (the direction the camera is facing) and the robot's y-direction is in the camera's *negative* x-direction. All of these orientations need to be taken into account when determining the robot's motion in the robot's frame of reference.

Chapter 5

Simulation and Control Interface

In this chapter, we present the details on vision-based tracking, teleoperation, and the motion control of the ER1 robot. For vision-based tracking, we describe the methods used to identify and track an object with known color and geometry by tracking the transformation matrix from a reference frame fixed on the object to the robot's frame of reference. We discuss the difficulties with the object tracking program and possible solutions to these issues. For teleoperation, we study the strategy of using single or multiple computers to relay the control from a remote command center. For the ER1 motion control, we provide information on wheel calibration for accurate position control.

5.1 Vision-Based Identification and Tracking

The object used in our experiment is a 2-D square-shaped object in red color. The features for this object are the edges and the corners. The object tracking algorithm begins by grabbing an image frame from the on-board camera. A Gaussian filter is implemented for noise reduction, followed by a color detection algorithm which marks the object as white and the background as black (refer to Fig. 5.2(b) and (c)).

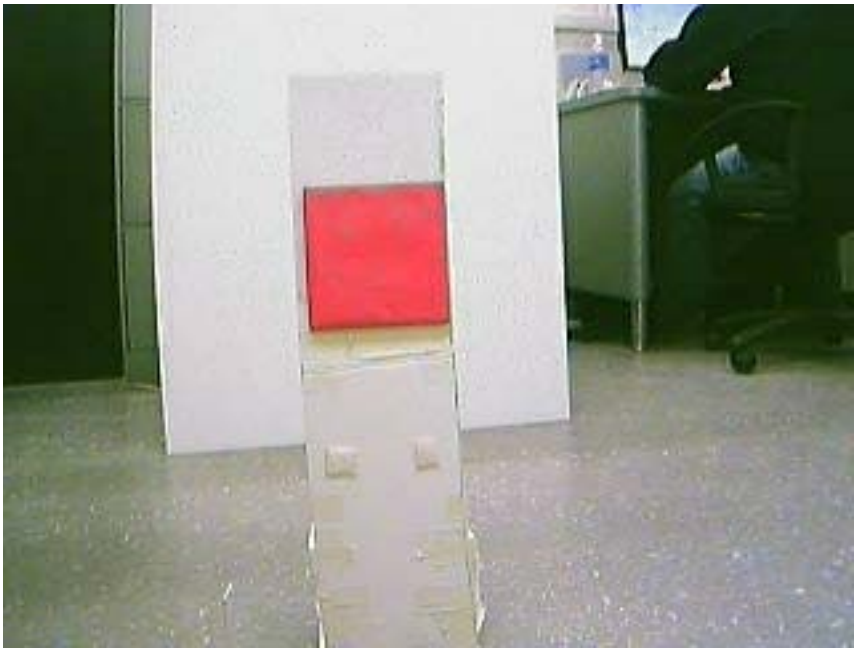


Figure 5.1 Object used for tracking from the on-board camera

The color detection algorithm detects the object by applying a ratio comparison between the red value and the green and blue values in the image (i.e. red/green, red/blue). If both ratios are within the ratio threshold values, the pixel is given the value of 1. Otherwise, the pixel is given the value 0.

After performing edge detection on this result (Fig. 5.2(d)), a linear curve fit of the upper and lower edges of the image is calculated. If the slope of the upper curve fit is determined to be too close to zero, the curve fits of the left and right sides of the image will be calculated inaccurately using the curve fit algorithm due to an infinite slope condition (i.e. vertical sides). The inaccurate curve fit results are displayed in Fig. 5.4. However, since we already know that the edge is vertical, we can use the average x-coordinate value for the left and right-hand sides of the image with a negligible change in accuracy of the actual vs. calculated corner locations.

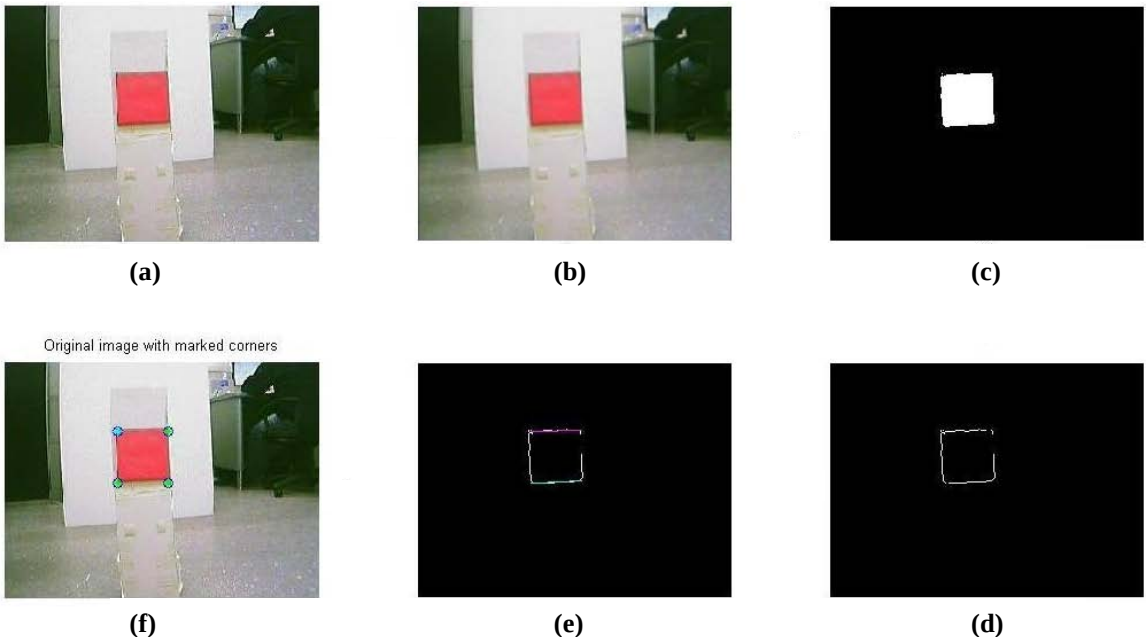


Figure 5.2 Determining the locations of the object: **(a)** original image; **(b)** filtered image; **(c)** color detection; **(d)** edge detection; **(e)** slope detection for upper and lower edge; **(f)** final corner locations

If the slopes of the upper and lower curve fits are determined to be outside of a zero slope threshold, a linear curve fit of the left and right sides of the object can also be calculated. By determining the intersections of these four curve fits, we can pinpoint the locations of the four corners of the object. The obtained corner locations (shown in Fig. 5.2(f) and Fig. 5.3(f)) are plugged into (3.10) as x_p and y_p to determine the object's location in both the robot and world reference frames.

Errors from the Vision Algorithm and Possible Corrections The calculation of the image corners did not always run smoothly. Several situations have resulted in consistent errors that needed to be fixed in the program. One such situation is shown in Fig. 5.4

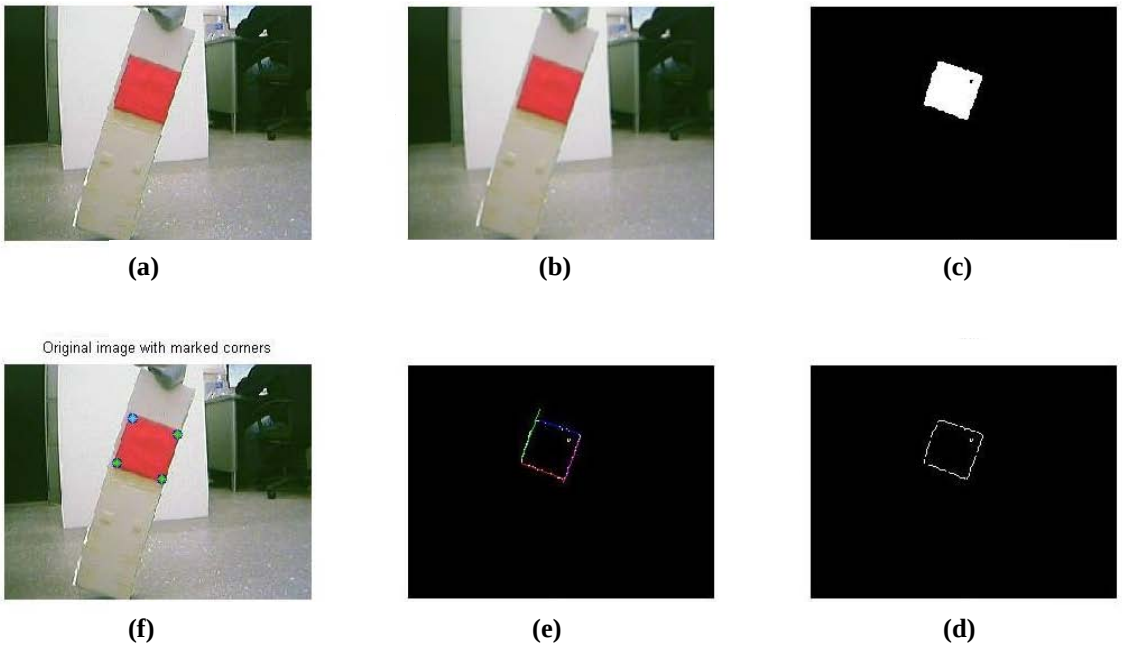


Figure 5.3 Determining the locations of the object: **(a)** original image; **(b)** filtered image; **(c)** color detection; **(d)** edge detection; **(e)** slope detection for each edge; **(f)** final corner locations from curve fit intersections.

The issue with this image is that the curve fits for the sides of the image were detected to be near infinite slopes. Matlab has trouble handling this situation and gives an incorrect response for the resulting curve fit. As mentioned previously, to remedy this issue (as mentioned previously), if the slope of the top and bottom edges of the curve iswithin a set threshold of zero slope, it can be assumed that the sides are vertical and we can set the x-coordinate value equal to the average value of the vertical edges. It is for this reason that the calculations of the slopes of the upper and lower edges are calculated prior to the sides.

Another error occurred for an image held at a roughly 45 degree angle to the horizontal. Since an angled image does not have an issue with curve fitting a vertical line, all four edges can be curve fitted (refer to Fig. 5.5). The error with the image that was held at a 45 degree angle is that, when calculating the curve fit, the left side of the

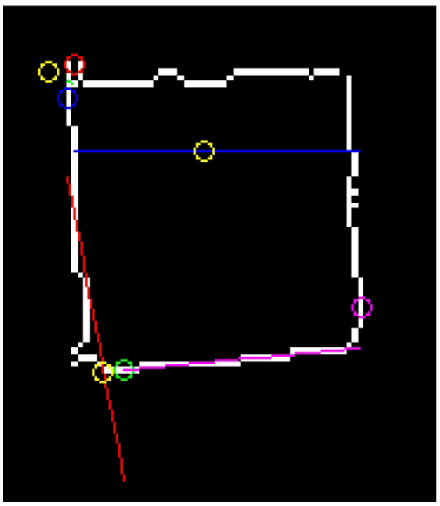


Figure 5.4 Error occurring for the calculation of infinite slope curve fit.

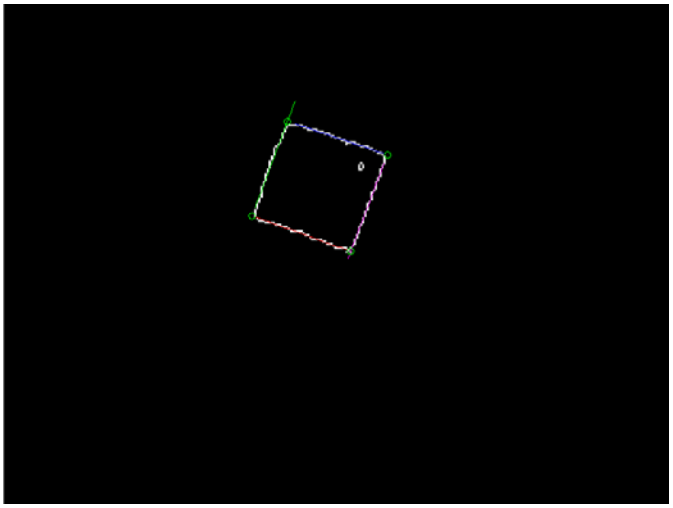


Figure 5.5 Angled image after curve fit. The green circles in the corners are the locations where the curve fits intersect.

maximum and minimum points along the y-axis would have opposite values to those on the right side. This would result in the slopes of the left side and right side to average out to zero slope and the resulting curve fit would be a horizontal line (refer to Fig. 5.6). To solve this issue, the curve fit was separated into two parts. By taking the average x-value to determine the change from positive to negative slope and finding the slopes of the left and right sides separately, we could then compare them and take the greater absolute value of the two for our curve fit. This method completely removed the slope miscalculation issue and gave more consistent readings. The results from the combination of the corner detection program and the extrinsic parameter calculation resulted in an accurate and consistent readout of the object location. These results can be seen in Fig. 5.7.

Before tracking of the object can begin, the object needs to be located by the robot. This task is required to be completed before any future motion is to be

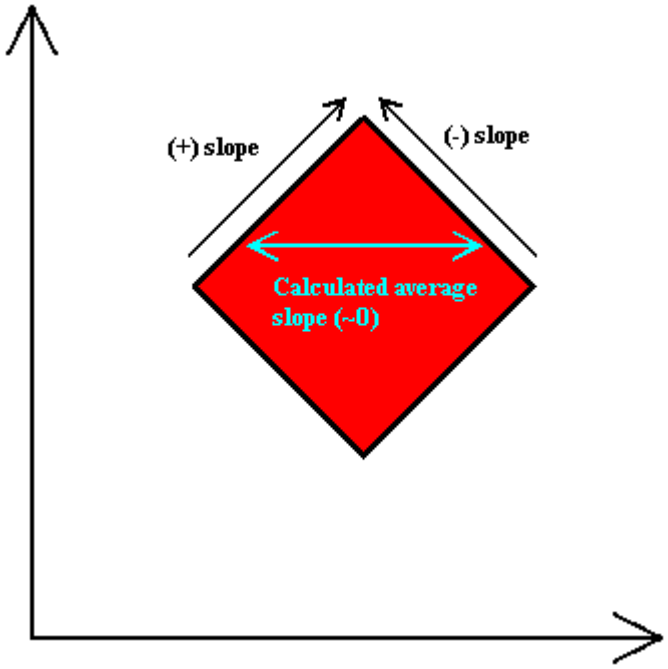


Figure 5.6 Slopes of left and right sides of object angled at 45 degrees averaging out to a zero-slope line.

implemented. The search for the object is a simple method. By turning the robot small increments followed by an image analysis of its current view, the robot can confirm the presence of the object. If the object is not in the field of view, the robot will continue to turn clockwise by a set increment and recheck its surroundings. Once the object is in view, the robot will center the object before determining whether the object is at a reasonable distance for tracking.

After the object has been centered, its distance from the robot is measured. If the distance of the object is not within an acceptable range for accurate corner detection, the robot will move to within an acceptable range. The importance of this adjustment is to increase the accuracy of the transformation matrix for the object tracking algorithm. If the object is located too close to the robot, the shadow on the surface of the object can be cause for an inaccurate distance reading. The farther away from the robot, the greater the

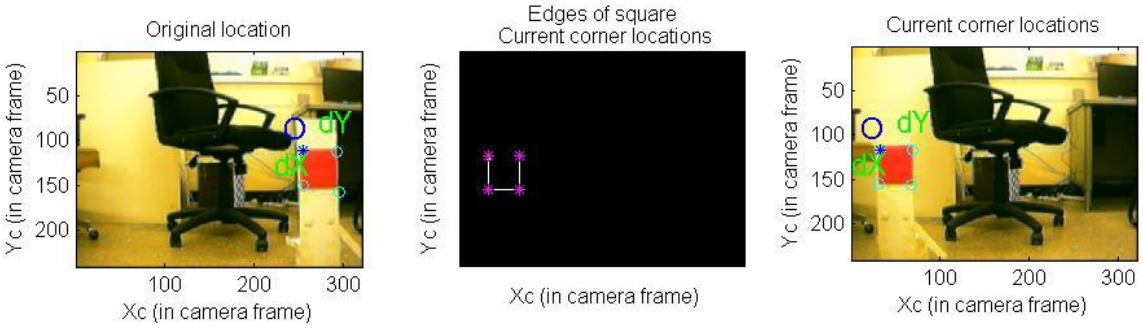
(a) **(b)**

Figure 5.7 Final results of corner location algorithm for **(a)** upright; **(b)** and angled images. Note: The object reference frame origin (marked as light blue) is always located at the corner location closest to the upper-left hand corner of the camera's field of view (origin of image frame of reference).

difficulty in determining accurate corner locations and the more precise the readings of the object corners would need to be. The acceptable range chosen for the object was between 75 and 100cm from the robot. After this adjustment, the object tracking algorithm is implemented.

Object Tracking The final and most important step for the robot is accurately tracking the object. By determining the motion of the object, the robot calculates a response for the robot to adjust itself by moving to a new position. This new position will view the object's new location at the same orientation as the original position.

In order to determine the motion required by the robot, an image of the desired object position needs to be taken. This image is the initial position of the object after distance adjustment. Using this image as a reference, the most current position of the object is constantly compared with the reference image to determine if the robot is required to change its current position (refer to Fig. 5.8). Upon a quick inspection of the



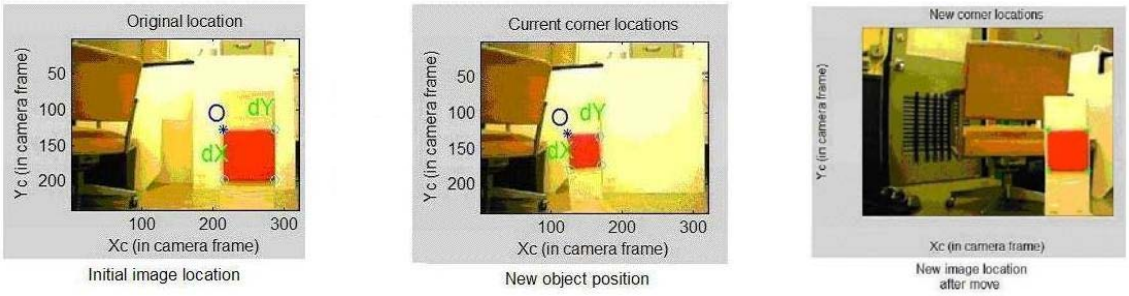
(a) **(b)** **(c)**

Figure 5.8 Image from the on-board camera for object tracking. **(a)** Original object location; **(b)** current object corner detection; **(c)** and current object location

motion from the reference image, coupled with the angle that the object was turned along its own x-axis, the motion required by the robot is calculated.

The determining factor for the robot motion is the net distance the robot would need to adjust itself. If the required translation from the robot's current position is at least 30cm, the robot adjusts itself to the new position with a new orientation (refer to Fig. 5.9). The robot would position its reference frame to be at a position that would change the transformation matrix from the robot to the object to be as close as possible to that of the initial image. It is at this point that trajectory tracking presented in section 2.3 is put to use.

Once the 30cm threshold is breeched, the x-y coordinates that the robot needs to reposition itself to is sent to the simulation algorithm. The algorithm takes these coordinates and performs trajectory tracking of a reference robot following an arced motion to the robot's calculated position. The returned values from the algorithm are the x-y coordinates of the robot for every projected x-y coordinate for the reference robot at each time step.



(a) **(b)** **(c)**

Figure 5.9 Results for object tracking. **(a)** initial object location; **(b)** new object location; **(c)** and object location after robot motion**.**

5.2 Teleoperation

ER1 can be controlled by using the teleoperation mode. The robot control software provides the ability to connect to the robot via the internet and control the robot remotely from any location. Through the command prompt, the ER1 can accept simple motion and vision detection commands, and generate feedbacks such as the robot position.

To demonstrate a scenario of the general teleoperation mode for ER1, we can use a three-computer setup that consists of a host computer, a remote computer, and an on-board computer. The remote computer sends commands to the host computer. The host computer, which acts as a liaison between the remote computer and tablet PC, forwards these commands to the tablet PC. The onboard computer performs the actual robot motion after receiving commands sent from the remote computer. In practice, it is more practical to use two computers where one is the command station and the other is the on-board computer. For our intended purposes, the simplest setup involves using a single computer which assumes the roles of both host and client.

For sending and receiving information packets through an internet connection, we use the *pnet*[1] and *ERToolkits*[2] toolboxes in Matlab. The *pnet* toolbox opens connections as a client or server to send/receive text strings. It is used for network communication with other applications & allows for remote Matlab control. The *ERToolkits* toolbox communicates with the RCC of the robot directly by using the *pnet* toolbox. By using the *pnet* and *ERToolkits* toolboxes to connect to the ER1, commands can easily be sent to the robot and feedback can be read from the encoders.

For the single computer set up, we use a laptop for both a remote or local connection. This enables control of the robot on the same computer that the RCC is being used on. By specifying the IP address to connect to as a local home address (127.0.0.1), we no longer rely on any external resources. This removes the need of a second or third computer. It also eliminates the need for an internet connection as well. This scenario allows us to achieve local control even though the RCC is running on teleoperative mode.

5.3 Motion Control of ER1

For straight-line motion prior to calibration, the robot would always undershoot the target distance. To remedy this issue, the robot was given a straight-line trajectory using several attempts to move a total distance of 200cm at set increments for each attempt. These increments were 4, 5, 10, 20, 50, 100, and 200cm. For each increment, a different number of iterations were performed to reach the 200cm mark (50 iterations for 4cm, 40

[1] The *pnet* toolbox can be obtained from http://www.mathworks.com/matlabcentral/fileexchange.
[2] The *ERToolkits* toolbox can be obtained from http://vision.ucsd.edu/mobile_robot.

Input Distance (cm)	25	25	25	25	25	25	25	25
Iteration	1	2	3	4	5	6	7	8
Average Stepsizes (cm)	22.67	22.73	22.60	22.77	22.73	22.77	22.63	22.77
Error (cm)	2.33	2.27	2.40	2.23	2.27	2.23	2.37	2.23
Final Distances (cm)	22.67	45.40	68.00	90.77	113.5	136.3	158.9	181.7

Table 5.1 Iteration values of 25cm stepsize.

iterations for 5cm, 20 iterations for 10cm, etc). The resulting positions for each iteration was then plotted to compare with the ideal final position. After the robot completed the commanded motion, the stepsize errors were compared with each other and the pooled standard deviation was calculated.

The average error in the actual stepsize of the robot was 2.33cm with a pooled standard deviation of 0.0693. Fig. 5.10 shows the plot of the averages along with the standard deviations for each set of tests. The resulting equation for correcting this error in position can be written as

$$D_{final} = D_{input} + 2.33 \quad \textbf{(5.1)}$$

where D_{input} is the desired distance for the robot to move and D_{final} is the actual value sent to the RCC.

For nonlinear motion, since the ER1 uses eye-in-hand configuration and because the position error is always miniscule, this error in position of nonlinear motion would be negligible in the final working model. In addition, any overshoot would be taken into account with every image capture the camera would perform. For this reason, it was unnecessary to calibrate the nonlinear motion of the robot.

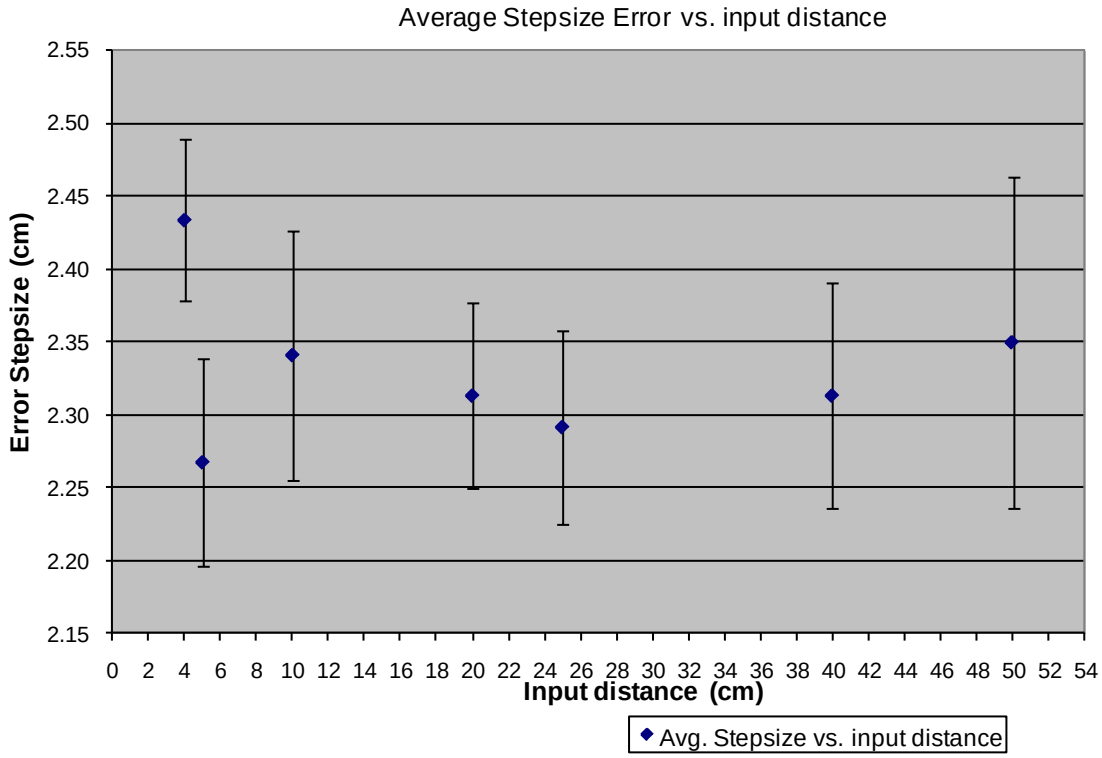


Figure 5.10 Plot of Average error of each step vs. the distance input to the ER1 and their standard deviations.

5.4 Discussion

In section 5.1, an issue with illumination had developed for object detection. This issue arose after determining the RGB threshold values for a particular room brightness. Originally, the color detection was calculated using the max and min RGB values rather than ratios. Because of this, if there was too much or too little background lighting after the threshold was set, the object location and orientation would not be accurately determined. This was due to the automatic brightness correction (refer to Fig. 5.11).

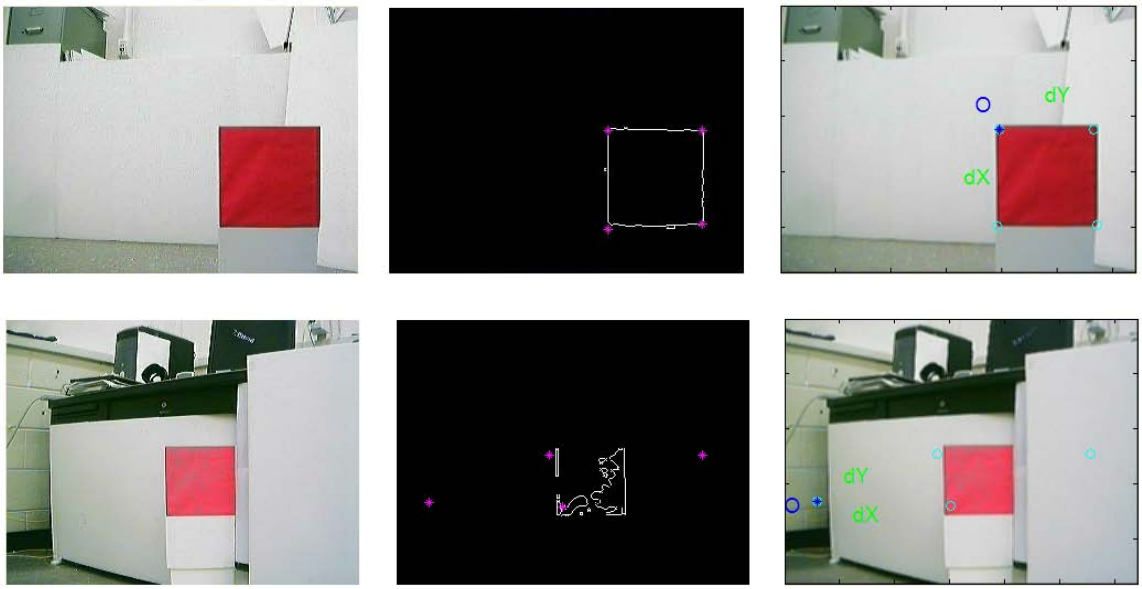
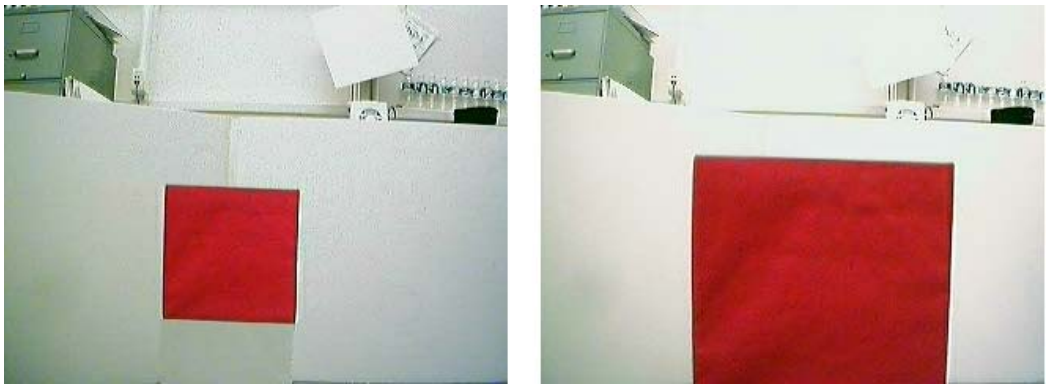


Figure 5.11 The first row shows the image with accurate corner detection. The second row shows an example with inaccurate corner detection due to the interference of automatic brightness correction.



(a) **(b)**

(c) **(d)**

Figure 5.12 Blue tinting of the webcam's view. **(a)** Original video; **(b)** Object is moved toward webcam; **(c)** Blue saturation begins; **(d) S**aturation remains.

Another issue that arose is the saturation level in the image. If the webcam was activated with the object too close, or if the object was brought too close to the webcam for a short amount of time (<10 seconds), the webcam's video and resulting snapshots would gain a bluish hue (refer to Fig. 5.12). This hue would cause the object to go undetected by the robot. The best solution that has been determined to address this issue is to avoid the robot from approaching close enough to the object to cause this hue saturation to occur.

Chapter 6

Integrated Simulation Environment

In this chapter, we present a general framework for integrated, hardware-in-the-loop simulation and control in mobile robot studies. We use the ER1 as the testbed and the vision-based object tracking as our test subject for the implementation environment. This is followed by an in-depth explanation of the capabilities and functions of the environment.

6.1 General Framework

The main objective for the integrated simulation environment is to combine all the individual components including the sensor information gathering (object identification), trajectory generation, and robot motion control. Firstly, the robot needs identify a given object of known dimensions. Secondly, the object's initial location relative to the robot is to be determined. Thirdly, the robot must be able to track the object's motion relative to its initial position. Lastly, the robot must be able to properly implement a control to track and maintain a constant relative position with respect to the object in order to accurately follow the reference trajectory.

6.2 Simulation Environment Capabilities

One of the main outcomes of the thesis research is a user friendly integrated simulation environment that allows direct implementation and testing of the various robot control functions (refer to Fig. 6.1). The simulation environment developed also grants the user a straightforward approach to implementation of the combined vision-based tracking of the object and backstepping control response. The object's relative motion is calculated by applying the object tracking algorithm presented in chapter 5 to obtain the robot's target position. Using the integrator backstepping method (chapter 2) combined

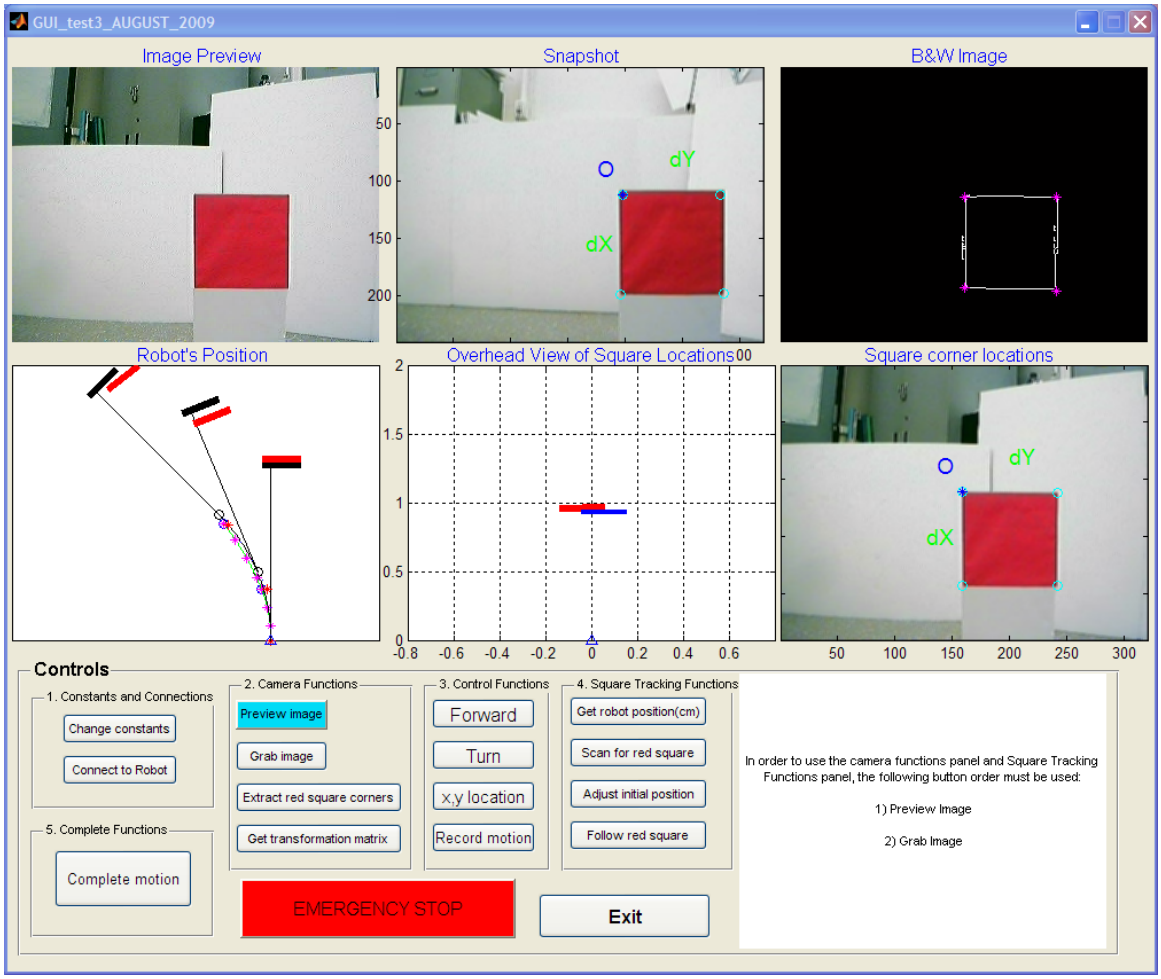


Figure 6.1 Simulation environment developed for control of the nonholonomic mobile robot.

with the Matlab toolboxes and ER1 teleoperation (chapter 5), the robot motion is implemented. This motion is a smooth transition from the robot's initial position to the robot's new target position.

6.3 Functionalities and Operations

Descriptions of the Graphical User Interface The *Image Preview* window displays the live video feed from the webcam. The *Snapshot* window displays a snapshot from the *Image Preview* window. This image is used as the reference image for object tracking. The *B&W Image* shows a black and white outline of the object. The *Object Corner Locations* window shows the calculated corners of the object with the object's frame of reference origin marked with a blue star. The *Overhead View of Object Location* window displays an overhead of the initial object location (i.e. the reference image as a blue line), the current object location relative to the initial object location (red line) and a blue triangle for the robot's location. Since this overhead view is to show the motion of the object, this view is shown from the camera's frame of reference. For this reason the blue triangle is always placed at (0,0).

The *Robot's Position* window shows multiple plots. Two plots are for the object's location and two are for the robot's motion. For the two object location plots, the first plot is the object's actual location in the world frame of reference. This is shown as a thick black solid line. This line has a thin black line extending to a point marked by a black circle. This black circle marks the ideal location where the robot should move to

in order to maintain a constant transformation matrix from itself to the object (i.e. ideal object tracking). The second plot, represented by a thick red line, is the object's initial location perceived by the robot in the world frame of reference. The robot's location for each of these perceived views of the object is marked by a red star (i.e. actual object tracking).

For the robot motion plots, the ideal path of the robot is shown as a thin black arc. The predicted path, which is calculated using the input from the object tracking algorithm, is shown as a thin green arc. The magenta stars are the points on the green arc that the robot is given to follow. For perfect object following, the black curve and the green curve should overlap identically. (The plot of the object's ideal location and the robot's ideal path motion are only used for the setup presented in section 6.4. These are used for testing purposes only. Otherwise, these plots can be omitted.)

The controls are pushbuttons, each with a unique function. Each of these control buttons was placed into one of five control blocks for straightforward usability which are discussed in the following five subsections.

6.3.1 Constants and Connections Block

Change Constants The *Change Constants* button accesses the Matlab script file which contains all of the constants that are used by the robot. These constants include the intrinsic parameters of the webcam and the red/green, and red/blue ratios used for the object identification algorithm.

Connect to Robot The *Connect to Robot* button allows the user to connect to the robot from any IP address. Since the graphical user interface buttons are designed to work locally, after pressing *Connect to Robot* on a remote computer, the user would have to send commands using Matlab's command window rather than using the user interface buttons. This provides the user with the freedom to control the robot from any location provided that the user inputs the robot's controlling laptop's IP address.

6.3.2 Camera Functions Block

The *Camera Functions* block contains all of the buttons that are used to access the webcam and analyze snapshots taken of the robot's environment.

Preview Image The *Preview Image* button is used to display a live video feed from the webcam. The video feed is displayed in the *Image Preview* display window.

Grab Image The *Grab Image* button takes a snapshot from the video feed and displays the results in the *Snapshot* display window. The purpose of allowing the user to grab an individual image is to give the user the ability to visually inspect the image to determine whether or not to continue with the object identification algorithm.

Extract Object Corners This button determines the locations of the object's corners by using object identification. The image used is the instantaneous view in the *Image Preview* display window. After the object identification algorithm calculates the corner

locations of the object, the black and white outline and the original image of the object with marked corners are displayed in the *B&W Image* and *Object Corner Locations* display windows, respectively (refer to Fig. 6.1).

Get Transformation Matrix The calculation of the transformation matrix is completed during object identification (i.e. the transformation matrix is calculated once the *Extract Object Corners* button is pressed). The *Get Transformation Matrix* button displays the transformation matrix in the Matlab command window.

6.3.3 Control Functions Block

The Control Functions block contains all of the buttons used to control the motion of the robot for displacements inputted by the user.

Forward The *Forward* button allows the user to move the robot forward by a distance inputted by the user. Once this button is pushed, the user is prompted to enter a value. The robot moves forward by the value inputted by the user (in centimeters). To move the robot backwards, the user would place a negative sign in front of the desired distance for the robot to move.

Turn The *Turn* button allows the user to turn the robot clockwise by an angle inputted by the user. Once this button is pushed, the user is prompted to enter a value. The robot turns clockwise by the value inputted by the user (in degrees). To turn the robot

counterclockwise, the user would place a negative sign in front of the desired angle for the robot to turn.

X,Y Location The *X,Y Location* button allows the user to move the robot to specific x-y coordinates that are inputted by the user. Once this button is pushed, the user is prompted to enter x-y coordinates. The robot moves to the x-y coordinates inputted by the user (in centimeters) where the x-direction in the robot's frame of reference is forward, and the y-direction in the robot's frame of reference is to the left.

Record Motion The *Record Motion* button plots the x-y location of the robot in the *Robot's Position* display window. After pressing the *Record Motion* button, the user is prompted to enter an x-y location for the robot to approach. As the robot is moving, the x-y coordinates of the robot are plotted in real-time.

6.3.4 Object Tracking Functions Block

The Object Tracking Functions block contains all of the functions the robot uses to track the object. This block incorporates the capabilities of both the *Camera Functions* block and the *Control Functions* block and integrates them for object tracking.

Get Robot Position (cm) The *Get Robot Position (cm)* button displays the robot's current position (in centimeters) relative to the robot's starting position. The starting position is the location in which the ER1 RCC is first initialized and it is always marked

as the origin. After the robot moves away from this position, the user can reset the robot's position from within the RCC.

Scan for Object Once the *Scan for Object* button is pressed, the robot will begin scanning the room for the object. This is accomplished by turning clockwise a set increment, taking a snapshot of the robot's current view, and checking to see if the object is within this view via the object identification algorithm. If the object is not in view, the robot will continue scanning until the object is identified. Once the object is identified, the robot adjusts its position to center the object in the robot's field of view.

Adjust Initial Position The *Adjust Initial Position* button causes the robot to adjust the distance between itself and the object. If the robot is not between 75 and 100cm from the object, or if the robot does not see the object head-on, the robot will adjust its position to be within these two values by moving to the appropriate location. After the adjustment, the robot will be between 75 and 100cm from the object and will be facing the object head-on. This assures a more accurate reading of the object corner locations.

Follow Object The *Follow Object* button causes the robot to follow the object. The robot begins by using the object tracking algorithm in a nonstop loop. The initial image and current image comparisons are plotted in the *Overhead View of Object Location* plot along with a printout of the x-, y-, and angle differences in units of meters and degrees in the Matlab command window. Once a threshold value of 30cm is required for the robot's motion to maintain the initial transformation matrix relative to the object, the trajectory

tracking method described in section 2.3 is implemented and adjusts the robot's position accordingly.

Complete Motion The *Complete Motion* button runs the *Follow Object* function in a nonstop loop. This allows multiple motions of both the object and the robot to be plotted. This also provides a comparison of the object's ideal motion with the object's perceived motion by the robot. The *Complete Motion* function is provided as an example to future users of the capabilities of the integrated simulation environment.

6.4 Testing and Results

In order to test the nonholonomic robot motion, the *Complete Motion* function was tailored for the robot to move in a semicircular path with a radius of 91cm. The semicircular motion was split into nine identical segments for the robot to follow. Prior to each motion, the object was placed in set locations along the semicircular path to have the robot move to the correct location on the semicircle. The object's actual location in

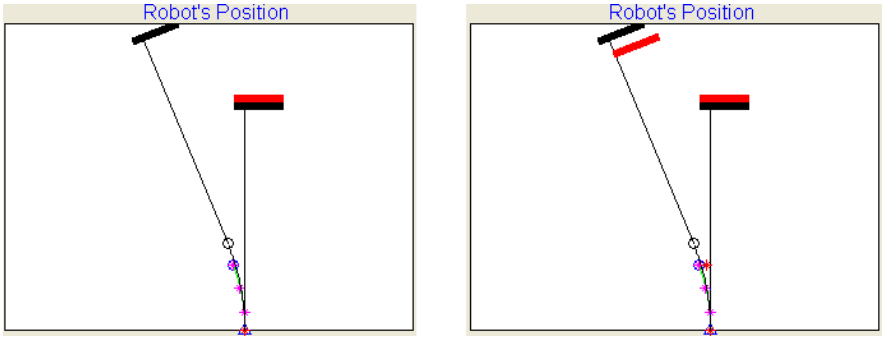


Figure 6.2 (a) Object's actual position, ideal path, and robot's perceived path along with; **(b)** the object's initial position relative to the robot's new orientation in the world frame of reference.

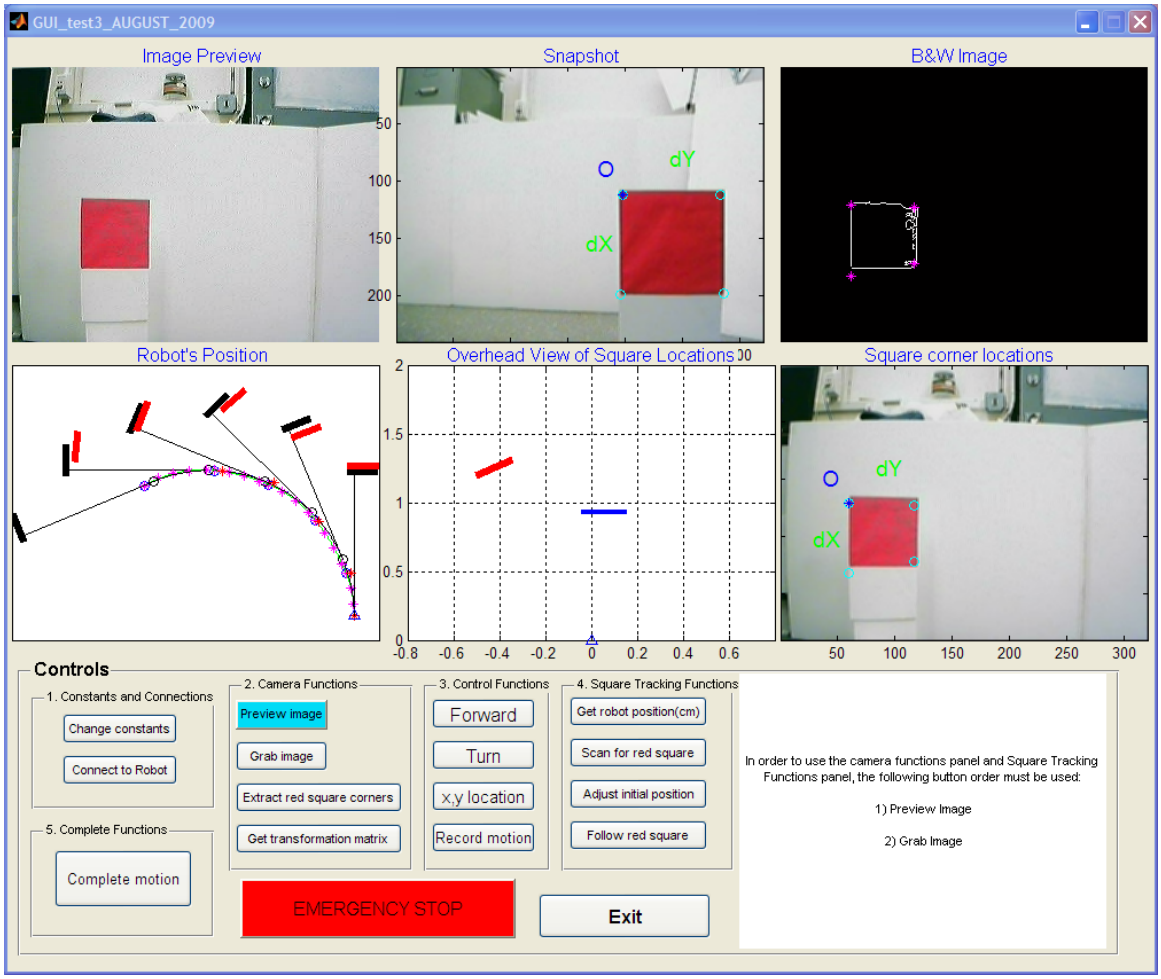


Figure 6.3 Simulation environment's view prior to the fifth motion segment of the robot.

the world frame of reference, along with the ideal path the robot should follow, was then plotted in the *Robot's Position* window. In addition, the perceived path calculated via trajectory tracking was also plotted. After the robot completed each perceived segment of the semicircle, the object's initial location relative to the robot's new location in the world frame of reference was plotted and compared to the object's actual location (refer to Fig. 6.2).

The results show that the robot accurately follows the actual semicircular path with very little offset. The difference between the ideal path and the actual path of the

robot was due to differences between the ideal location and the perceived location of the object by the robot. This offset was caused by inherent precision limitations in the object identification algorithm. The results for the first 5 semicircular segments are shown in Fig. 6.3.

6.5 Discussion

There are several advantages of using the integrated simulation environment presented in this chapter. The graphical user interface functions are designed to be intuitive and easy to learn. The simulation environment offers a front-end between the researcher and RCC software for object tracking and sensor based controller design. This includes determining the object's relative orientation to the robot in either the robot's reference frame or in the world reference frame. The user is also given the ability to freely modify the graphical user interface's functions to fit one's own research needs. In addition, the robot control is not limited to only using the backstepping method, but can be replaced with any other method preferred by the user. For data handling, the simulation environment developed here establishes the ability to store data to be analyzed. The RCC provided for the ER1 does not store any data to be either viewed or analyzed other than predefined images where the user must input the object's relative distance from the robot. The environment presented in this chapter has the capability to store all of the information the robot obtains. This information includes the initial and current views of the robot and the relative location of the object and robot in both the world and robot

frames of reference. All of this information can be viewed or accessed through the user interface.

To use this simulation environment and the graphical user interface, the ER1 RCC must be running and set to listen to incoming API requests through a port. Once the RCC is set to accept connections to the same port as that set in Matlab, the user interface can then be initialized.

The *Connect to Robot* function should be used for remote access only. Otherwise, this function is not needed since the localhost (127.0.0.1) is used by default.

Chapter 7

Conclusions and Future Work

The objective of this thesis is the design and integration of a simulation environment and an ER1-based experiment platform to aid the research in mobile robot control. Along the way, we studied a trajectory tracking control scheme using ER1 robot and implemented vision-based object tracking algorithms using the on-board camera. Simulation and experimental results show that the overall design is feasible and easy to use. All the components in the design have been tested with a certain level of success, including image capturing, object identification and tracking, communication between the graphical user interface and the ER1's RCC, as well as robot motion control. The environment designed in this thesis will provide researchers with an excellent tool for studying the capabilities of the mobile robot systems, and more importantly, for designing and testing new sensor-based control schemes.

Our results show that the current vision algorithm we implemented is sensitive to the change of the illumination of the object or in the environment. Conditions such as too much reflection on the objects, not enough light, or shadows from the surroundings would interfere with the robot's perception of the object. Extensive research has been done in the computer vision community along the direction of increasing the contrast and the robustness of object identification and tracking. For future work, we will look into the integration of these algorithms into our simulation environment.

References

[1] A. Yamashita, M. Fukushi, J. Ota, T. Arai, and H. Asama, "Motion Planning for Cooperative Transportation of a Large Object by Multiple Mobile Robots in a 3D Environment," in *IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000, pp. 3144-3151.

[2] D. F. Hougen, M. D. Erickson, P. E. Rybski, S. A. Stoeter, M. Gini, and N. Papanikolopoulos, "Autonomous Mobile Robots and Distributed Exploratory Missions," in *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems,* 2000, pp. 221-230.

[3] K. Singh and K. Fujimura, "Map Making by Cooperating Mobile Robots," in *IEEE International Conference on Robotics and Automation*, 1993, pp. 254-259.

[4] D. E. Franklin, A. B. Kahng, and M. A. Lewis, "Distributed Sensing and Probing With Multiple Search Agents: Toward System Level Landmine Detection Solutions," *Detection Technologies for Mines and Minelike Targets*, vol. 2496, pp. 698-709, 1995.

[5] J. L. Baxter, E. K. Burke, J. M. Garibaldi, and M. Norman, in *Autonomous Robots and Agents Series: Studies in Computational Intelligence*. vol. 76: Subhas Mukhopadhyay, Gourab Sen Gupta, 2007.

[6] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative Search and Rescue with a Team of Mobile Robots," in *8th International Conference on Advanced Robotics*, Montrey, CA, 1997, pp. 193-200.

[7] L. E. Parker, "Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems," *J. Physical Agents*, vol. 2, pp. 5-14, 2008.

[8] R. Siegwart and I. R. Nourbakhsh, "Introduction to Autonomous Mobile Robots", 2004.

[9] R. Fierro and F. L. Lewis, "Control of a Nonholonomic Mobile Robot: Backstepping Kinematics into Dynamics," in *Proceedings of the 34th IEEE Conference on Decision and Control*, New Orleans, LA, 1995, pp. 3805-3810.

[10] S. Hutchinson, G. D. Hager, and P. I. Corke, "A Tutorial on Visual Servo Control," *IEEE Transactions on Robotics and Automation*, vol. 12, p. 19, 1996.

[11] R. Shapiro, "Direct linear transformation method for three-dimensional cinematography," *Research Quarterly*, vol. 49, p. 8, 1978.

[12] J. Heikkila and O. Silven, "A Four-step Camera Calibration Procedure with Implicit Image Correction," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997, pp. 1106-1112.

[13] J. Heikkila and O. Silven, "Calibration Procedure for Short Focal Length Off-the-Shelf CCD-Cameras," in *13th International Conference on Pattern Recognition*, Vienna, Austria, 1996, pp. 166-170.

[14] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab," Pasadena, CA, 2008.

[15] "ER1 Robot Navigation Project," The Artificial Perception Laboratory, 2005.

[16] M. A. Abidi and T. Chandra, "Pose Estimation for Camera Calibration and Landmark Tracking," in *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, pp. 420-426.

[17] A. M. Bloch, M. Reyhanglu, and N. H. McClamroch, "Control and Stabilization of Nonholonomic Dynamic Systems," *IEEE Transactions on Automatic Control*, vol. 37, p. 11, November 1992.

[18] D. C. Brown, "Close-range Camera Calibration," *Photogrammetric Engineering*, vol. 37, p. 11, 1971.

[19] Z. Dodds, S. Santana, B. Erickson, K. Wnuk, J. Fisher, and M. Livianu, "Teaching Robot Localization with the Evolution ER1," Harvey Mudd College, Claremont2004.

[20] R. Fierro and F. L. Lewis, "Control of a Nonholonomic Mobile Robot Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 9, p. 11, July 1998 1998.

[21] T. Fukao, H. Nakagawa, and N. Adachi, "Adaptive Tracking Control of a Nonholonomic Mobile Robot," *IEEE Transactions on Robotics and Automation*, vol. 16, p. 7, October 2000.

[22] T. Hu and S. X. Yang, "An Efficient Neural Controller for a Nonholonomic Mobile Robot," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Alberta, Canada, 2001, pp. 461-466.

[23] E. R. Inc., *ER1 User Guide*. Pasadena, CA, 2002.

[24] O. Jesorsky, K. J. Kirchberg, and R. W. Frischholz, "Robust Face Detection Using the Hausdorff Distance," in *Audio- and Video-Based Biometric Person Authentication*. vol. 2091: Springer Berlin / Heidelberg, June 2001, pp. 90-95.

[25] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A Stable Tracking Control Method for an Autonomous Mobile Robot," in *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, pp. 384-389.

[26] M.-S. Kim, J.-H. Shin, S.-G. Hong, and J.-J. Lee, "Designing a Robust Adaptive Dynamic Controller for Nonholonomic Mobile Robots under Modeling Uncertainty and Disturbances," *Mechatronics*, vol. 13, p. 12, 2003.

[27] G. Klančar and I. Škrjanc, "Tracking-error model-based Predictive Control for Mobile Robots in Real Time," *Robotics and Autonomous Systems*, vol. 55, p. 9, June 2007 2007.

[28] Y. Koren and J. Borenstein, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," in *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, CA, 1991, pp. 1398-1404.

[29] J.-C. Latombe, "Robot motion planning". Boston: Kluwer Academic Publishers, 1991.

[30] N. J. Nilsson, "Principles of Artificial Intelligence". San Francisco: Morgan Kaufmann Publishers Inc., 1980.

[31] Y. Ono, "Corridor Navigation of a Mobile Robot using a Camera and Sensors--Multi-Agent Approach," in *Graduate Faculty of The University of Georgia*. vol. Masters Athens: The University of Georgia 2003, p. 80.

[32] Y. Ono, H. Uchiyama, and W. Potter, "A mobile robot for corridor navigation: a multi-agent approach," in *Proceedings of the 42nd annual Southeast regional Conference*, Huntsville, Alabama, 2004, pp. 379-384.

[33] P. Song and V. Kumar, "A Potential Field Based Approach to Multi-Robot Manipulation," in *IEEE International Conference on Robotics and Automation*, Washington DC., 2002, pp. 1217-1222.

[34] C. W. Warren, "Multiple Robot Path Coordination Using Artificial Potential Fields," in *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, pp. 500-505.

[35] J. Weng, P. Cohen, and M. Herniou, "Camera Calibration with Distortion Models and Accuracy Evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 14, p. 25, October 1992 1992.

[36] Y. Yamamoto and X. Yun, "Coordinating Locomotion and Manipulation of a Mobile Manipulator," in *Preceedings of the 31st Conference on Decision and Control*, Tucson, AZ, 1992, p. 6.

[37] Z. Zhang, "Flexible Camera Calibration by Viewing a Plane from Unknown Orientations," in *IEEE International Conference on Computer Vision*, 1999.

[38] Z. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, p. 4, 2000.