UNIVERSITY OF CALIFORNIA, SAN DIEGO

Robot Traffic School: Improving Autonomous Navigation in EOD Robots

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Thomas Denewiler

Committee in charge:

Professor Thomas R. Bewley, Chair
Professor Raymond de Callafon
Professor Ryan Kastner

2011

The thesis of Thomas Denewiler is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2011

DEDICATION

To Grandma Denny,
it's a small step from board games to Kalman filters,
To my parents,
for their time and encouragement in everything.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The enthusiasm and energy of Professor Thomas Bewley has been an inspiration and his support and advice have been invaluable.

I am grateful to Gideon Prior, Nima Ghods, Amin Rahimi and Steve Stancliff for our many long conversations while learning how to build better robots.

I would also like to thank Mike Bruch, Bart Everett and the ACS team (Gaurav Ahuja, Donnie Fellars, Greg Kogut and Brandon Sights) as well as Jason Lum, Kelly Grant and the EOD technicians at SPAWAR for their support with both hardware and software.

And without the love and encouragement from Silvie Georgens I would not have made it this far.

ABSTRACT OF THE THESIS

Robot Traffic School: Improving Autonomous Navigation in EOD Robots

by

Thomas Denewiler

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California, San Diego, 2011

Professor Thomas R. Bewley, Chair

Advancements in the autonomous navigation of robots increases the range of behaviors that can be implemented, consequently increasing the utility of the robots to end users. To achieve these advancements, the state estimation and controls algorithms for Explosives Ordinance Disposal (EOD) robots have been studied and improved. In this work, I integrated a high precision, differential GPS system to measure ground truth positions, which were then used to find more accurate system and measurement noise covariance values. The more accurate noise models improved the state estimate of an extended Kalman filter. Independently, a model-based control law was implemented for a vehicle with nonholonomic unicycle constraints kinematics using a Lyapunov method. The Lyapunov controller was implemented on several different EOD robots and is compared to the previously

existing PID controller with respect to navigation near simulated obstacles and in open space. Practical considerations for tuning the Lyapunov controller design variables are explored, and recommendations are given for several operating scenarios. The improved algorithms were implemented using multiple different robots. The algorithms are currently running on EOD robots used in the field. This work will accelerate development of advanced maneuvers, such as retroverse over long distances as well as obstacle avoidance.

Chapter 1

Introduction

In this chapter, we give a brief overview of the research performed for this thesis and the contributions this research provides for the robotics community. Additionally, an introduction to autonomous navigation and its relevance to end users is provided.

1.1 Thesis Outline & Contributions

The problem of autonomous navigation is not isolated to any one technical area. Instead, it is a combination of estimation, controls and planning. Estimating the robots position, and its environment, is required in order to determine where the robot is located in the world. The area of controls is concerned with which commands will cause a robot to move from its current location to a desired location. Planning involves determining the best path to get a robot to a desired location and includes issues such as obstacle avoidance.

This research focuses on the estimation and controls aspects of autonomous navigation for Explosives Ordinance Disposal (EOD) robots. The estimation algorithm uses a Kalman filter to determine the robots location in the world. The Kalman filter was improved in two separate ways: (i) Several bugs in the Kalman filter implementation used on the EOD robots were found and fixes were applied so that the equations are calculated correctly. The result of these fixes dramatically improved the state estimate of the robot. Details are in Section 3.3. (ii) Discriminative training of the covariance matrices describing the noise models

used by the Kalman filter was used to find more accurate covariance values. The new noise models resulted in an improved state estimate provided by the Kalman filter. Details are in Section 3.5.1.

The original controls algorithm on the EOD robots used a Proportional-Integral-Derivative (PID) controller that required a large amount of time to tune so that it would work on different driving surfaces at variable speeds. This thesis describes the implementation of a model-based controller that uses a kinematic model of a nonholonomic vehicle with unicycle constraints, based on a Lyapunov method. The new controller is a direct replacement for the PID controller. The Lyapunov controller is shown to work at variable speeds with very little tuning. Additionally, the Lyapunov controller provides more free design variables than the PID controller. These design variables can be used to shape the trajectory of the robot as it moves from its current location to a desired location and they are not available with the PID controller. Results of testing the Lyapunov controller while modifying the design variables are shown and guidelines for setting these values are provided. The Lyapunov controller is described in Chapter 4.

Background information is provided in Chapter 2. Chapter 3 describes the Kalman filter and how it is used to estimate the state of the robot. The PID and Lyapunov controllers are described in Chapter 4. Results that show improvements in the estimation and controls algorithms are described in Chapter 5. Finally, the conclusion and suggestions for future work are described in Chapter 6.

The main contribution of this research is that the EOD robots investigated here have greater autonomy which allows for less human oversight of basic functionality. The algorithms developed during this reasearch have been implemented on fielded, production systems, and are shown to work better than the original algorithms.

1.2 The Need for Autonomy

Robots have been developed to assist humans in tasks that are generally considered dirty, dangerous or boring. Recently, robots have found a useful niche as a tool to help EOD teams assess and eliminate threats from improvised explosive

devices (IEDs), commonly referred to as roadside bombs. The use of robots allows humans to maintain a safe stand-off distance while investigating a scene. Clearly, this falls under the dangerous category. The current method that EOD teams use involves teleoperation of the robot from the base of operations to the object of interest. The teleoperation task consumes the operators energy and focus while they are navigating the robot to and from a goal location. During this process the operator is exposed and vulnerable to external threats including ambushes and enemy snipers.

As technologies mature they provide humans with better tools. However, as the use of robots increases shortcomings are discovered, such as the vulnerability due to teleoperation, and that opens up avenues for improvements. One approach to reducing the amount of work humans are required to perform is to give the robots more intelligence via autonomous behaviors. This is accomplished using additional sensors, specialized actuators and more advanced software to automate as many routine tasks as possible. In time, more complex tasks such as navigation over rough terrain, around obstacles and back to a home location will become routine and automated as well, thus freeing up the operators to focus on higher level tasks.

When adding autonomy to robots nearly all of the tasks can be summarized by the following questions:

- Where am I?
- What's around me?
- Where do I want to go?
- How do I get there?

The initial attempt at adding autonomy to EOD robots resulted in somewhat erratic driving behavior. This was especially evident near obstacles, as the robot trajectory would not be smooth while changing speeds when attempting to move around the obstacle [Bruch 00]. In this thesis we will look at smoothing out the trajectories taken by the robot by making improvements to the state estimation (Where am I?) and controls (How do I get there?) algorithms. This work ignores

actual obstacle detection (What's around me?) and will be using a simple planning algorithm (Where do I want to go?) to simulate obstacles in the robot path. The simulated obstacles will force the robot to change direction and speed multiple times. One of the benefits of a new controls algorithm with respect to planning will be discussed as well.

Other tasks for small robots include sending them into buildings that are dangerous due to structural damage or unknown, possibly hostile elements inside. The goal is to have the robots map the interior of the building and provide images so the operator can assess any danger prior to humans entering the buildings [Congress 06]. After the attacks on the World Trade Center on September 11, 2001, several small robot systems were used to look for survivors in the rubble and to help assess structural damage to nearby buildings [Everett 02].

Speed, efficiency and precision are valuable characteristics that can result in the success or failure of a mission. Better autonomous navigation algorithms can improve upon these characteristics in small robot systems. An example of an advanced behavior that becomes possible with better navigation is the ability to have a robot retrotraverse, meaning to find its way back to a preset location without human intervention. The benefits of improved navigation include less time waiting for a robot to get close to an IED or clearing a building. This results in less time for humans in a hostile or dangerous environment. In search and rescue situations this would lead to less time for searching and more time for rescuing.

This work is especially relevant for the dirty, dangerous and boring tasks where robots are most useful because it allows humans to focus their concentration, time and effort on being clean, safe and efficient.

Chapter 2

Background

This chapter gives background information about the robots used for the experiments, the OCU used to communicate with the robots, the software running on the robots and the area used for testing.

For the robots to drive around on their own all they require is an estimate of where they are, a path to follow, a controller to determine the actuator outputs and motor controllers to perform the controller outputs.

2.1 Small Unmanned Ground Vehicles

There are two commonly used robots in EOD applications, the iRobot Packbot and Kinetiq Talon, which are differential drive robots with two motors that drive tracks on either side of the robot. These two robots were used in the experiments that follow. Additionally, the Urbot (developed by SPAWAR Systems Center, Pacific (SSCPAC)) and the Autonomous Solutions Chaos robots were tested. The OCU and software developed for estimation, planning and controls runs on all four robots although most of the testing was done using a Packbot. Note that differential drive, skid steer and unicycle-like robots are synonymous descriptions in the literature and used interchangeably in this thesis.

Each of the robots has a computer that accepts teleoperation commands consisting of a linear and angular velocity. Additionally, each robot has a payload bay where another computer and sensors used for autonomous navigation are located. The payload computer is used to read sensor data and calculate appropriate linear

and angular velocity commands that are sent to the robot computer. The standard sensors available on each robot are a gyro, wheel encoders, inertial measurement unit (IMU) and a global position system (GPS) receiver.

The Packbot is manufactured by iRobot and can be seen in Figure 2.1.

Figure 2.1: iRobot Packbot

The Talon is manufactured by Kinetiq and is shown in Figure 2.2.

Figure 2.2: Kinetiq Talon

The Urbot is an experimental prototype of a small robot developed by SSCPAC and is shown in Figure 2.3.

Figure 2.3: SSCPAC Urbot

The Chaos is manufactured by Autonomous Solutions and is shown in Figure 2.4.

Figure 2.4: Autonomous Solutions Chaos

2.2 MOCU & JAUS

The Multi-Robot Operator Control Unit (MOCU), shown in Figure 2.5, is a highly configurable front-end for simultaneous command and control of multiple systems and was created at SSCPAC [Powell 08]. MOCU has the ability to use a variety of communications protocols for interfacing to different systems and uses the Joint Architecture for Unmanned Systems (JAUS) protocol to send and receive data to all of the robots used in this research [Rowe 08]. A combination of teleoperation using a joystick controller and autonomous navigation were used to collect data and test new ideas for estimation and controls. From within MOCU waypoints can be drawn on an overhead image of the operating area and from those waypoints a route is generated and downloaded to the robot using the JAUS protocol. The robot will then attempt to drive that route autonomously and send back status information to MOCU using the controls and estimation code that is the focus of this research.

2.3 Sensors

The Packbot and Talon have their own computers that take in commands for desired linear and angular velocities. Those computers then output the correct motor controller commands that cause the robot to move at the desired velocities. Typically, the desired linear and angular velocity commands are generated by a user with a remote control. The research in this thesis was concerned with generating the velocity commands autonomously, or without human input, using a set of

Figure 2.5: Controlling Packbot with MOCU

sensors installed in one of the payload bays of the robot.

For most of the results presented here the computer used for running the estimation and controls algorithms is the Beckhoff CB4051 with an Intel 2.0GHz Core 2 Duo CPU. Tests were also conducted using an Intel Atom CPU as the payload computer with very similar results as obtained with the Core 2 Duo CPU.

The IMU in the payload bay is a Microstrain 3DM-GX1 that outputs Euler angles and angular rates.

Two different GPS receivers were used on the robots. The GPS receiver used most often on all of the robots is a Novatel OEMV. Additionally, a uBlox NEO-6Q GPS receiver was used on both the Talon and Packbot during the course of experimentation. The Novatel receiver performed better overall, though the uBlox was adequate. All of the results presented here use data from runs with the Novatel receiver.

A gyroscope is used to measure angular velocity of the robots. The KVH DSP-3000 Fiber Optic Gyro is installed in the payload bay of all the robots tested here.

Originally, a compass was not used on the Packbot, but after initial testing it was determined that the heading reported by the Microstrain 3DM-GX1 was not very reliable and an Ocean Server OS5000 compass was added to the payload bay. This compass gives outputs for pitch, roll and yaw angles. It was determined that

the yaw angle output of this compass was not reliable enough to be used very often. Instead, a combination of integrating the KVH gyro output and, when available, the GPS yaw angle output, was used to generate the yaw angle estimate.

The Packbot and Talon are manufactured with wheel encoders that are read by the main computer and used to calculate the linear and angular velocities of the robot. The payload computer has access to these values when communicating with the main computer. Initially, the wheel encoder data was not being used so that backwards compatibility with other small robots without wheel encoders would be maintained. However, the systems without wheel encoders are no longer used by EOD groups, so the wheel encoder data has been incorporated into the sensor suite used by the algorithms in this thesis.

2.4 Estimation and Control Software

The SSCPAC robotics group has developed the Autonomous Capabilities Suite (ACS) which incorporates many different robotics related algorithms into a single software package that can be run on a wide variety of robots. ACS is able to easily accomodate different payload and sensor suites [Ahuja 06]. The Kalman filter implementation was done in ACS.

A JAUS library written by SSCPAC provides for communications between the robots and MOCU.

The original path planning and controls software was written by SSCPAC for the Man Portable Robotic Systems (MPRS) program [Bruch 02] and extended for use on unmanned surface vehicles [Ebken 05], [Larson 06], [Larson 07]. The new model-based controller is implemented in the MPRS code and will be ported to ACS in the near future.

A single program was built by combining the ACS Kalman filter code, the JAUS library and the MPRS code. This is the program that was run for all of the experiments presented here.

Chapter 3

State Estimation

One of the ACS libraries is the extended Kalman filter, which is used on the EOD robots for state estimation and is the main method used to answer the question "Where am I?". The idea behind the Kalman filter is relatively straightforward in that the robot has some basic idea of where it is in the world using its sensors, but there is uncertainty involved in that estimate due to:

- different measurement accuracies from the sensors,
- multiple sensors measuring the same state and giving slightly different values,
- some states that are not measured,
- imperfect models of the robot dynamics.

The Kalman filter is a method to merge physical models and sensor data to realize an estimate of robot location, what its orientation is and how fast it is moving. This estimate is better than any of the individual sensor measurements [Simon 06], [Grewal 08], [Orderud 05]. There are four models used in the Kalman filter:

- system model based on physics,
- measurement model to transform sensor output to state coordinate system,
- system noise model,
- measurement noise model.

3.1 State Space Models

Kalman filters and modern control systems (see Chapter 4) use the idea of a multi-dimensional state space to encapsulate all of the relevant information that is known about a system. In the case of robots like the ones used in these experiments the states of interest are position, orientation and linear and angular velocities. In general, a system is described by nonlinear equations that describe how the state variables change through time and how measurements of the system are related to the states as given by

$$\begin{aligned} \dot{x} &= f(x, u, t) \\ \dot{y} &= h(x, t). \end{aligned} \tag{3.1}$$

The state variables are given in vector form by x and the sensor measurements are contained in the vector y. The state space equations are a means of representing, with compact notation, how the state of a system changes through time based on the initial state of the system plus the inputs to the system, u. This representation allows the trajectory (or motion through time) to be calculated. The inputs are assumed to include any external forces applied to the system, as well as actuation provided by the system itself. The function $h(x, t)$ transforms the sensor measurements into state variables.

For the robots used in these experiments the state vector used follows that found in [Kelly 94a], [Kelly 94b], where the state variables are

$$x_k = \begin{bmatrix} x & y & z & V & \theta & \phi & \psi & \omega \end{bmatrix}^T.$$

In this vector, x, y and z are positions, V is linear velocity, θ, ϕ and ψ are Euler angles (corresponding to pitch, roll and yaw) and ω is angular velocity, as shown in Figure 3.1.

3.2 The Kalman Filter

The ACS Kalman filter is typical of all Kalman filters in that it consists of a prediction update step and a measurement update step. The prediction update is

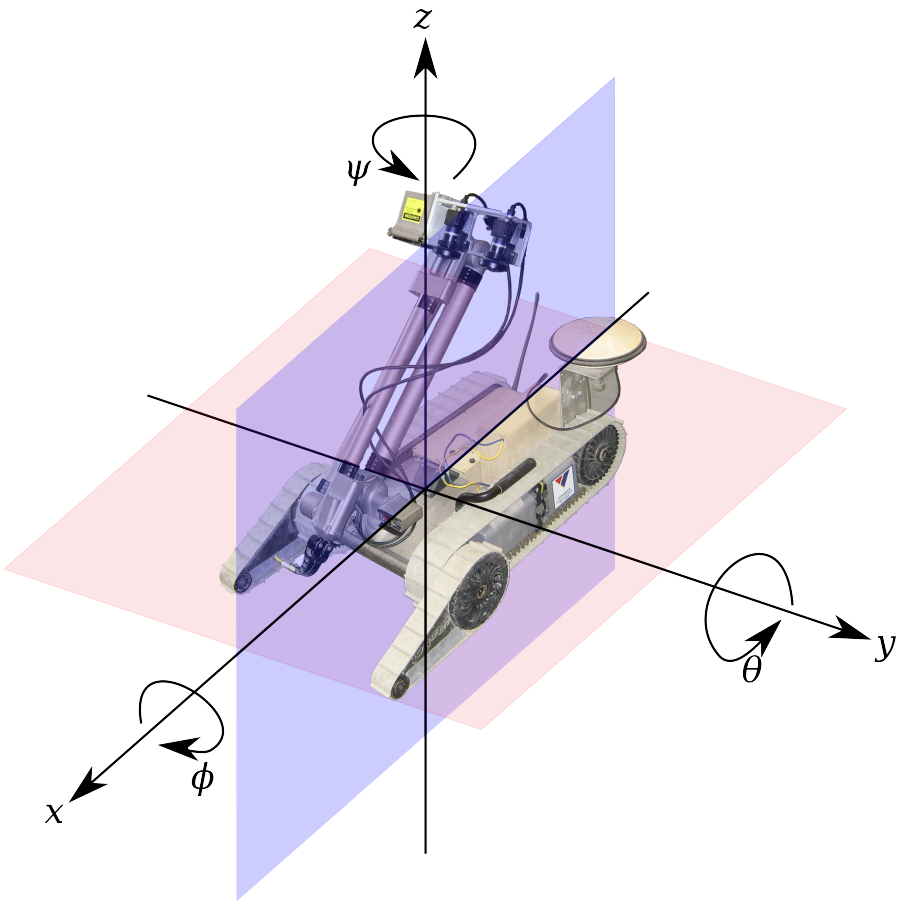


Figure 3.1: Packbot Axes for Position and Orientation

run as fast as possible and the measurement update is run whenever new sensor data becomes available, as depicted in the block diagram of Figure 3.2. The prediction update step uses a model of the system dynamics (or kinematics) and a measurement of elapsed time to determine where the system is in the world. This system model will inevitably have errors. Some of the errors are due to effects that are not captured in the model, including (im)precision of the clock on the computer for measuring time. Other errors are due to simplifying assumptions that are made in order to calculate the equations of the system model in real-time using embedded computers. The measurement update step is basically a feedback step to help correct for errors in the system model using sensors to provide current data [Kelly 94a].

The Kalman filter that runs on the small robots used in these experiments is implemented on a digital computer and is necessarily used in discrete time

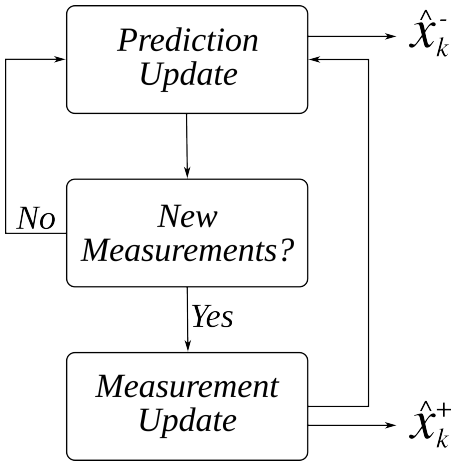


Figure 3.2: The Kalman Filter Algorithm

rather than continuous time because of the nature of computers. Additionally, the standard Kalman filter equations make the assumption that the system and measurement models are linear. From [Kelly 94a], [Simon 06] the discretized and linearized versions of (3.1), giving the state space equations, are represented as

$$x_{k+1} = \Phi_k x_k + w_k$$
$$y_k = H_k x_k + v_k.$$

Here, Φ_k is the discrete time system model (or state transition matrix) relating the state at time $k+1$ to time k in the absence of inputs or noise and it models the system dynamics. The term w_k is system noise due to an imperfect model. H_k is the measurement matrix which relates the measurements to the state vector and v_k is the sensor measurement noise.

The ACS Kalman filter uses an unforced model so all inputs are considered disturbances to the steady-state dynamics. This means that w_k includes noise, modeling errors, external forces and actuator forces. The fact that all inputs are modeled as disturbances could lead to a large source of error in the estimate of the state variables. However, [Kelly 94a] explains that very little is known about the powertrain dynamices of most autonomous vehicles, so any control-input model has a good chance of being wrong, and the use of that type of model could exagerrate rather than reduce errors.

The prediction update step marches the system dynamics forward in time

using the equations

$$\begin{aligned} \hat{x}_{k+1}^{-} &= \Phi_k \hat{x}_k^{+} \\ P_{k+1}^{-} &= \Phi_k P_k^{+} \Phi_k^T + Q_k \end{aligned} \tag{3.2}$$

and the measurement update step provides feedback from sensor data using the equations

$$\begin{aligned} K_k &= P_k^{-} H_k^T \left[H_k P_k^{-} H_k^T + R_k \right]^{-1} \\ \hat{x}_k^{+} &= \hat{x}_k^{-} + K_k \left[y_k - H_k \hat{x}_k^{-} \right] \\ P_k^{+} &= \left[I - K_k H_k \right] P_k^{-} . \end{aligned} \tag{3.3}$$

P_k is the state covariance matrix and K_k is the Kalman gain. The noise models are $Q_k = E[w_k w_k^T]$ and $R_k = E[v_k v_k^T]$. Q_k is the process covariance matrix giving a measure of the expected noise in the system model and R_k is the measurement covariance matrix giving the expected noise of the sensors. Using (3.2) and (3.3) a state estimate of the robot can be obtained at any time. This estimate can be used for the controls algorithms, to give feedback to an operator or to share with other systems that can change their behavior based on the robot state.

The variables $\hat{x}_k^{-}$, P_{k+1}^{-} in (3.2) refer to the mean and covariance of the estimate obtained after the prediction update step (as denoted by the negative superscript) and $\hat{x}_k^{+}$, P_k^{+} in (3.3) refer to the mean and covariance of the estimate after the measurement update step in the Kalman filter (as denoted by the positive superscript). Each of the terms in the Kalman filter equations is discussed in more detail in subsequent sections.

3.2.1 Models in the Kalman Filter

The Kalman filter relies on several models that are used to determine the mean and covariance of the estimated states for a system. The ACS Kalman filter uses four such models:

- System dynamics model Φ_k,
- measurement model H_k,

- system noise model Q_k,
- measurement noise model R_k.

It is important to recognize that the fidelity of and interaction between these four models determines how well the Kalman filter output will represent the true state of the robot.

3.2.2 Assumptions in the System Model

It is nearly impossible to develop models that completely capture all of the attributes of most systems, including robots that are expected to operate in many different physical environments. For this reason assumptions are made to simplify the system model. The following assumptions allow the prediction update step to be calculated in a reasonable amount of time using modern computers so that a state estimate is available that the control systems can act upon in real time. The assumptions also allow a single system model to be abstracted and used on multiple similar but different robotic vehicles such as those described in Section 2.1.

Low Dynamics Assumption

The first assumption made is that, from one time step to the next, the robot will not be accelerating fast enough in any direction for the sensors on the robot to be able to measure those accelerations. This means that the two velocities, linear and angular, in the state vector are assumed to be constant. The benefit of this assumption is that there are six fewer states that must be tracked in the state vector, one for acceleration about each axis of the robot.

Principal Motion Assumption

The second assumption says that, during a single time step, the position of the robot will only be a function of linear velocity and the orientation of the robot will only be a function of the angular velocity. Figure 3.1 helps in visualizing the effect of rotating the robot about its center and how that will not affect the position of the robot. Similarly, translation of the robot along the x, y or z axes

will not change the orientation of the robot. This assumption allows several terms in the system model to be set to zero.

3.2.3 Continuous to Discrete Time Transform

The system model will initially be developed using continuous time nonlinear differential equations, but the Kalman filter on the robots will be running on digital computers, so the model will need to be converted to discrete time. In continuous time the system model is

$$\dot{x} = Fx$$

and in discrete time the system model is

$$x_{k+1} = \Phi_k x_k.$$

The transformation from continuous to discrete time obeys an exponential matrix transformation with a Taylor series approximation given by [Gelb 74]

$$\Phi_k = e^{F\Delta_T} = I + F\Delta_T + \frac{(F\Delta_T)^2}{2!} + \ldots + \frac{(F\Delta_T)^n}{n!}$$

since

$$e^{F\Delta_T} = \sum_{i=0}^{N} \frac{(F\Delta_T)^i}{i!}$$

and $(F\Delta_T)^0 = I$, the identity matrix [Phadke 99].

A first order Taylor series approximation is generally considered good enough when Δ_T is small and when the nonlinearities in the system are small enough, meaning that the Taylor remainder is small. In the case of the robots used in these experiments that is assumed to be the case so the transformation used is simply

$$\Phi_k = I + F\Delta_T. \tag{3.4}$$

Note that when the low dynamics assumption and the principal motion assumption

are invoked the term $F^2 = 0$ so the nonlinearities are indeed small when those assumptions hold.

3.2.4 System Dynamics Model

A model of the system dynamics is necessary in order to propagate the state of the system forward in time in the absence of measurements. It is impossible to create a perfect model of the system, and even a nearly perfect model of the system will likely be too complex to compute fast enough for it to be useful. The best result that is typically available is a model with a large amount of assumptions, where the most important aspects of the dynamics are retained in the simplified model.

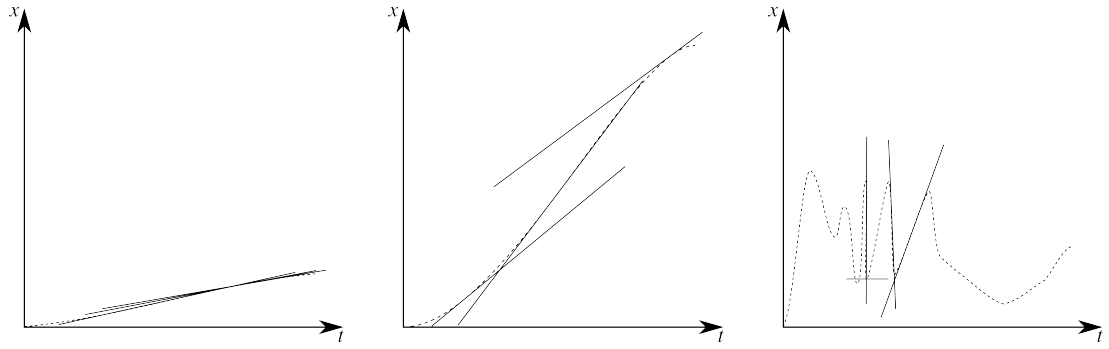
A nonlinear, continuous time model based on robot kinematics is developed in the body frame coordinate system using the states from Section 3.1. The governing differential equations are given by

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix} = \begin{bmatrix} V\cos\psi\cos\theta \\ V\sin\psi\cos\theta \\ -V\sin\theta \\ 0 \\ -\omega\sin\phi \\ \omega\tan\theta\cos\phi \\ \omega\cos\phi/\cos\theta \\ 0 \end{bmatrix}. \tag{3.5}$$

3.2.5 Extended Kalman Filter

The basic Kalman filter makes the assumption that both the system model contained in Φ_k and the measurement model in H_k are linear. The extended Kalman filter (EKF) allows for nonlinear models, such as that given by (3.5), to be used for Φ_k by linearizing the model around the state estimate. (Similar linearization can be applied to the measurement model if it is nonlinear.) To linearize the model the Jacobian, or matrix of partial derivatives, is determined and evaluated at the current state estimate. The idea of linearization of a nonlinear model is shown

in Figure 3.3, where the linearization can be seen to be more accurate when the system nonlinearities are small, meaning that the dynamics are slow.



(a) Slow dynamics. (b) Medium dynamics. (c) Fast dynamics.

Figure 3.3: Effects of Linearizing a Nonlinear Model

The nonlinear state dynamics were described in (3.5). The Jacobian of that state vector equation is calculated using

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial y} & \frac{\partial \dot{x}}{\partial z} & \frac{\partial \dot{x}}{\partial V} & \frac{\partial \dot{x}}{\partial \theta} & \frac{\partial \dot{x}}{\partial \phi} & \frac{\partial \dot{x}}{\partial \psi} & \frac{\partial \dot{x}}{\partial \omega} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial y} & \frac{\partial \dot{y}}{\partial z} & \frac{\partial \dot{y}}{\partial V} & \frac{\partial \dot{y}}{\partial \theta} & \frac{\partial \dot{y}}{\partial \phi} & \frac{\partial \dot{y}}{\partial \psi} & \frac{\partial \dot{y}}{\partial \omega} \\ \frac{\partial \dot{z}}{\partial x} & \frac{\partial \dot{z}}{\partial y} & \frac{\partial \dot{z}}{\partial z} & \frac{\partial \dot{z}}{\partial V} & \frac{\partial \dot{z}}{\partial \theta} & \frac{\partial \dot{z}}{\partial \phi} & \frac{\partial \dot{z}}{\partial \psi} & \frac{\partial \dot{z}}{\partial \omega} \\ \frac{\partial \dot{V}}{\partial x} & \frac{\partial \dot{V}}{\partial y} & \frac{\partial \dot{V}}{\partial z} & \frac{\partial \dot{V}}{\partial V} & \frac{\partial \dot{V}}{\partial \theta} & \frac{\partial \dot{V}}{\partial \phi} & \frac{\partial \dot{V}}{\partial \psi} & \frac{\partial \dot{V}}{\partial \omega} \\ \frac{\partial \dot{\theta}}{\partial x} & \frac{\partial \dot{\theta}}{\partial y} & \frac{\partial \dot{\theta}}{\partial z} & \frac{\partial \dot{\theta}}{\partial V} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \phi} & \frac{\partial \dot{\theta}}{\partial \psi} & \frac{\partial \dot{\theta}}{\partial \omega} \\ \frac{\partial \dot{\phi}}{\partial x} & \frac{\partial \dot{\phi}}{\partial y} & \frac{\partial \dot{\phi}}{\partial z} & \frac{\partial \dot{\phi}}{\partial V} & \frac{\partial \dot{\phi}}{\partial \theta} & \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial \psi} & \frac{\partial \dot{\phi}}{\partial \omega} \\ \frac{\partial \dot{\psi}}{\partial x} & \frac{\partial \dot{\psi}}{\partial y} & \frac{\partial \dot{\psi}}{\partial z} & \frac{\partial \dot{\psi}}{\partial V} & \frac{\partial \dot{\psi}}{\partial \theta} & \frac{\partial \dot{\psi}}{\partial \phi} & \frac{\partial \dot{\psi}}{\partial \psi} & \frac{\partial \dot{\psi}}{\partial \omega} \\ \frac{\partial \dot{\omega}}{\partial x} & \frac{\partial \dot{\omega}}{\partial y} & \frac{\partial \dot{\omega}}{\partial z} & \frac{\partial \dot{\omega}}{\partial V} & \frac{\partial \dot{\omega}}{\partial \theta} & \frac{\partial \dot{\omega}}{\partial \phi} & \frac{\partial \dot{\omega}}{\partial \psi} & \frac{\partial \dot{\omega}}{\partial \omega} \end{bmatrix}}_{F} \begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix}$$

which results in

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & c\psi c\theta & -Vc\psi s\theta & 0 & -Vs\psi c\theta & 0 \\ 0 & 0 & 0 & s\psi c\theta & -Vs\psi s\theta & 0 & Vc\psi c\theta & 0 \\ 0 & 0 & 0 & -s\theta & -Vc\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\omega c\phi & 0 & -s\phi \\ 0 & 0 & 0 & 0 & \omega c\phi/c^2\theta & \omega t\theta s\phi & 0 & t\theta c\phi \\ 0 & 0 & 0 & 0 & \omega s\theta c\phi/c^2\theta & -\omega s\phi/c\theta & 0 & c\phi/c\theta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{F} \begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix}. \tag{3.6}$$

Applying the principal motion assumption allows us to set the terms relating position and angular velocity to zero. Additionally, the terms relating orientation and linear velocity are set to zero. Finally, converting from continuous to discrete time to put ones on the diagonal (c.f. (3.4)) gives

$$x_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & c\psi c\theta\Delta_T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & s\psi c\theta\Delta_T & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -s\theta\Delta_T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -s\phi\Delta_T \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & t\theta c\phi\Delta_T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & c\phi\Delta_T/c\theta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Phi_k} \begin{bmatrix} x \\ y \\ z \\ V \\ \theta \\ \phi \\ \psi \\ \omega \end{bmatrix}_k. \tag{3.7}$$

This is the system model that is used in the prediction update step of the Kalman filter equations to calculate the state of the robot in between measurements.

Use of the extended Kalman filter requires that the system model be calculated at each time step using the most recent estimates of the state variables. This is due to the fact that partial derivatives of the nonlinear model in (3.5) are evaluated at the current state during the linearization process. Since the estimates are not calculated prior to operation of the robots, the recursive nature of the Kalman filter is required.

3.2.6 Measurement Model

The measurement model converts sensor data to the state variable coordinate system, and for a standard sensor suite is by far the simplest model used in the ACS Kalman filter. The idea is that, for a sensor that measures one of the states, the data output by the sensor has to have the same units and be in the same range as the state variable. This is best illustrated by an example.

Suppose a compass and an IMU both measure the yaw state variable. Let the compass output be $0 - 360°$, the IMU output be $0 - 2\pi$ $rads$ and the yaw state in the Kalman filter be tracked in the range $0 - 2\pi$ $rads$. If, for one time step, both compass and IMU measurements are available to be incorporated into the Kalman filter then the measurement update step from (3.3) is run and the H_k matrix is

$$H_k = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{180°} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and the y vector is a 2×1 column vector with the compass measurement in the $y_{(1,1)}$ element and the IMU measurement in the $y_{(2,1)}$ element.

There is one row per sensor measurement and one column per state in the measurement matrix, so in this example H_k is a 2×8 matrix. The compass measurement corresponds to the first row and the data is converted from degrees to radians. The IMU measurement corresponds to the second row and, since the units of the sensor data are the same as the units of the state variable, the data is not transformed at all.

In the case when there is a single sensor measuring each of the states, and those sensors output data that is in the correct units and range of the state variables, then $H_k = I$, the identity matrix.

3.2.7 Noise Models

The noise models attempt to estimate noise coming from the terms w_k and v_k in the equations

$$x_{k+1} = \Phi_k x_k + w_k$$
$$y_k = H_k x_k + v_k.$$

The Kalman filter uses two different noise models, one for system noise w_k and one for measurement noise v_k. The system noise model attempts to capture all of the effects due to an imperfect system model, linearization of the model, the low dynamics assumption and the principal motion assumption. The measurement noise model tries to compensate for sensors that do not measure the environment perfectly.

Noise in the Kalman filter is assumed to be zero mean with a Gaussian distribution. The models are set up as covariance matrices that describe the variance of the Gaussian distribution of each state and each sensor. When the noise in the system is *not* Gaussian then the Kalman filter is the best linear estimator and performance will degrade gracefully as non-Gaussian noise enters the system [Simon 10]. As noted in [Anderson 79], when dropping the Gaussian assumption all of the Kalman filter equations will propagate exactly the same for the mean and covariance of the states and measurements except that any information about higher order moments is lost, and the probability density functions cannot be reproduced. This means that the Gaussian assumption is valid when the higher order moments do not have a large effect on the distributions. When the higher order moments are significant then the Kalman filter performance will degrade gracefully and still be the optimal linear filter.

3.2.8 Kalman Gain

The Kalman gain matrix is used in the measurement update step (3.3) and essentially weights whether the state estimate will rely more on the system model

or the measurements. The gain matrix K_k is given by

$$K_k = P_k^- H_k^T \left[H_k P_k^- H_k^T + R_k \right]^{-1}$$

where

$$P_{k+1}^- = \Phi_k P_k^+ \Phi_k^T + Q_k.$$

From these equations it can be seen that all four models used in the Kalman filter are used to calculate the Kalman gain matrix, since Φ_k, Q_k, H_k and R_k are all present. In the measurement update step the Kalman gain is multiplied by the difference between the measured values y_k of the state and the expected values of those measurements based on the system model $\hat{y}_k$. Note that $\hat{y}_k = H_k \hat{x}_k^-$ is what the sensors would report if there were no noise in the measurements and the system model were perfect. The state estimate from the measurement update step is

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \left[y_k - H_k \hat{x}_k^- \right].$$

If the sensor data matches the system model prediction then the innovations term $y_k - H_k \hat{x}_k^- = 0$ and the Kalman gain is ignored. The innovation term is rarely, if ever, zero in practice. The interesting question then becomes whether to trust the noisy sensors or the imperfect system model to determine the state estimate. A large gain value will adjust the previous estimate from the prediction update step by a larger amount which means that the sensor data is weighted more heavily than the system model. Conversely, a small gain value will cause the estimate from the measurement update step to trust the system model more than the sensor data. It is important to note that the Kalman gain matrix K_k can have large values for some states (and sensors) and small values for other states (and sensors).

As discussed in Section 3.2.6, the measurement model is fairly simple, so the Kalman gain is really a function of the system model and the noise models. If the system model is a very good approximation to the real system then the system noise model should reflect that by having small values in Q_k. If the sensors are very good then the measurement noise model will have small values in R_k. Thus, the

Kalman gain matrix tends to take on values relative to the ratio of the elements in the Q_k and R_k noise models. From this it can be seen that developing accurate noise models is very important to the performance of the Kalman filter.

When the models are not set up correctly, then the measurements will likely not match the predicted state estimates. This will cause the Kalman gains to grow larger during each measurement update step to force the state estimate to trust the measurements more than the system model.

3.3 Fixing Bugs in ACS Kalman Filter

The ACS Kalman filter contained three major bugs in the equations and only worked because of a reforumlation of the standard Kalman filter equations. The first bug concerned the calculation of the Kalman gain matrix (see Section 3.2.8). The K_k matrix was being calculated correctly, but at the end of the calculation gains from previous sensors were added back into their former element locations in the matrix. This bug was fixed by removing the code that added previous gain elements back into the Kalman gain matrix. One of the consequences of this bug was that the calculation of the covariance of the state estimate in the prediction update step used the continuous time system model F (3.6) instead of the discrete time system model Φ_k (3.7). The result was such that instead of $P_{k+1}^- = \Phi_k P_k^- \Phi_k^T + Q_k$ (c.f. (3.2)) the Kalman filter used

$$P_{k+1}^- = F P_k^+ F^T + Q_k.$$

The discrete time system model Φ_k was used in the calculation of the state estimate $\hat{x}_k^-$ though. After this fix Φ_k was used appropriately in the state covariance calculation during the prediction update step.

The second bug was a consequence of calculating the Kalman gain incorrectly. The yaw angle calculation was not being done correctly since the state covariance estimate was wrong, so a correcting term was introduced that fixed the yaw angle estimate. This worked when a standard set of sensors were used since they reliably gave measurements at a fixed rate and affected the Kalman gain matrix in a consistent manner. This bug was also fixed by simply removing the correcting

term and the yaw angle was estimated correctly after the Kalman gain matrix was implemented properly.

The third bug was slightly less significant but still posed several problems. In the measurement update step (3.3), the measurement model H_k was set up to use the latest received data from each sensor at each time step rather than only use new measurements if they were available from each sensor. This bug manifested itself most prominently in sensors with slow update rates such as GPS. One particular robot configuration used the uBlox GPS receiver that output position fixes at $1Hz$, which resulted in the Kalman filter measurement update (running at $100Hz$) using the same position measurement for 100 time steps. That particular measurement should only have been used a single time out of the 100 time steps. This became obvious when the control system exhibited the behavior of cycling through slowing down and speeding up in a very jerky fashion. This occurred because the Kalman filter was trying to push the position estimate backwards toward the most recent GPS measurement instead of letting the system model Φ_k allow the robot position to propagate forward based on the robot kinematics.

3.4 Establishing Ground Truth

Quantitatively evaluating the performance of Kalman filters can be accomplished in several ways, the best of which is to analyze the output of the Kalman filter against ground truth. Although it is nearly impossible to establish ground truth over a large area in practice, the closer the measurements are to an absolute position in the world the better. To determine ground truth for the robots in these experiments a differential GPS (DGPS) system was used independently of the sensors on the robot so that very accurate measurements of the robots actual position could be logged. The ground truth measurements were then used in a post-processing step to determine how well the Kalman filter estimate corresponds to ground truth.

The DGPS system consists of a GPS receiver and serial radio that make up the base station, and a GPS receiver, serial radio and small computer that make up the roaming station as in Figure 3.4. The GPS receivers are both Novatel RT2

receivers using the Real Time Kinematics algorithm. The base station is located in a static position and is then configured to use a fixed position. The difference between the fixed position is compared to what the current position would be if it were not fixed. The difference between the fixed position and the calculated position are used to generate corrections that would put the position of the GPS antenna at the fixed position and those corrections are sent to and applied at the roaming station. This results in a standard deviation of 2 cm for the position output of the roaming station. The errors are due to the effects of the GPS signal passing through the atmosphere from the satellites to the antenna, as well as to multipath effects closer to the ground. The DGPS system is bootstrapped to the robot during testing runs to log data at a rate of $10Hz$ and is only used as a tool to measure ground truth for position, not to be used during normal operation.

With a highly accurate estimate of ground truth established, it becomes possible to not only study the performance of the Kalman filter, but to also begin determining whether to focus efforts in improving the autonomous navigation behaviors of the robots via the estimation or controls algorithms.

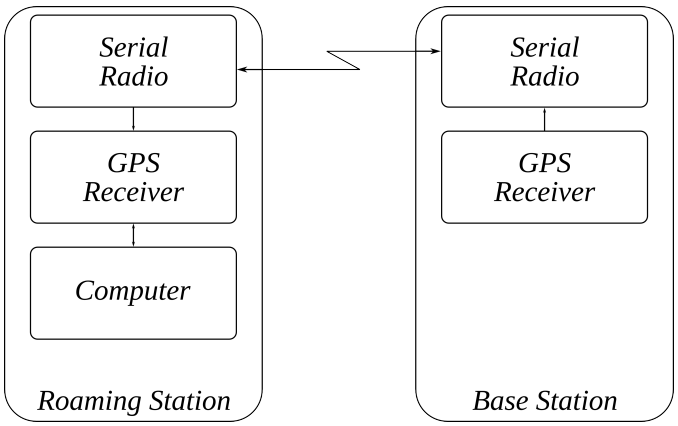


Figure 3.4: Differential GPS System Diagram

3.5 Identifying Noise Models

Attempting to determine the proper values for the noise models Q_k in (3.2) and R_k in (3.3) can be a laborious process and is often considered more of an art than a science, with engineer experience being a critical factor. In Section 3.2.7

it was shown that the Kalman filter performance is heavily dependent upon the accuracy of the noise models even though they are difficult to determine.

3.5.1 Discriminative Training of Kalman Filter Parameters

A method to automatically learn what the covariance matrices Q_k and R_k should be, using a discriminative training algorithm, was described in [Abbeel 05]. Building upon that work, [Sakai 10] recently used a variation of the algorithm described in [Abbeel 05] to learn parameter values for the noise filters used in their unscented Kalman filter. This method takes advantage of ground truth measurements obtained using a DGPS system like that described in Section 3.4. Note that in the following expressions the term $h(\mu_t)$ is the position output by the Kalman filter and y is the DGPS position output, so the goal is to find values of Q_k and R_k that minimize the difference between the output of the Kalman filter and DGPS ground truth.

There are several metrics that can be applied to the data, with the two most relevant metrics being the residual error metric and the prediction likelihood error metric. The residual prediction error is used to estimate Q_k and R_k using

$$\langle R_{\text{res}}, Q_{\text{res}} \rangle = \arg\min_{R,Q} \sum_{t=0}^{T} ||y_t - h(\mu_t)||_2^2.$$

When the covariance matrix P for the ground truth sensor is *not* a multiple of the identity matrix I then this metric is

$$\langle R_{\text{res}}, Q_{\text{res}} \rangle = \arg\min_{R,Q} \sum_{t=0}^{T} (y_t - h(\mu_t))^T P^{-1} (y_t - h(\mu_t)).$$

The error metric used for the residual prediction error method is

$$e = \left(\frac{1}{T} \sum_{t=1}^{T} ||h(\mu_t) - y_t||^2 \right)^{1/2}. \tag{3.8}$$

The prediction likelihood method use the metric

$$\langle R_{\text{pred}}, Q_{\text{pred}} \rangle = \arg\max_{R,Q} \sum_{t=0}^{T} -\log|2\pi\Omega_t| - (y_t - h(\mu_t))^T \Omega_t^{-1} (y_t - h(\mu_t))$$

where $\Omega_t = H_t \Sigma_t H_t^T + P$. The error metric used for the prediction likelihood method is

$$e = -\frac{1}{T} \sum_{t=1}^{T} \left(\log|2\pi\Omega_t| - (y_t - h(\mu_t))^T \Omega_t^{-1} (y_t - h(\mu_t)) \right) \tag{3.9}$$

where Ω_t is the same as in the above description.

3.5.2 Simulating Robot Runs

For the training algorithm to work the sensor data collected from a robot has to be replayed a large number of times while making small adjustments to the noise models. To accomplish this a simulation of the robot that runs faster than real time helps to reduce the amount of time taken to replay the data. As shown in Table 3.1, the position errors as calculated using (3.8) were very close between the simulated runs and the actual robot run. The average error was 0.0556 m over nine different runs, which gives a baseline for the convergence criterion.

Table 3.1: Kalman Filter Simulation Errors

Run	Actual Error (m)	Simulation Error (m)	Error Difference (m)
1	0.452886	0.423056	0.029830
2	0.177620	0.210472	0.032852
3	0.109484	0.080400	0.029084
4	0.494375	0.491243	0.003132
5	0.068306	0.035943	0.032363
6	0.000028	0.105102	0.105074
7	0.017564	0.033453	0.015889
8	2.599853	2.427348	0.172505
9	0.020686	0.100696	0.080010

3.5.3 Coordinate Ascent Algorithm

In order to evaluate the error metrics in (3.8) and (3.9) it is necessary to search through the space of parameters to find the optimal parameters to use in the Q_k and R_k matrices. Initially, a program was written to attempt a naive, brute force approach of trying every possible combination of parameters. The brute force algorithm turned out to be difficult to implement, took several days to run and returned unimpressive results that did not really improve the Kalman filter estimates. Following the description in [Abbeel 05] a coordinate ascent algorithm was then written with much better results and faster convergence times.

The coordinate ascent algorithm was initialized by a set of hand-tuned noise model parameters and then the robot run was simulated. For each parameter in the noise models the value of the parameter is then increased by $\alpha_i\%$ and the error metric is evaluated. If the resulting error is less than the previous best result the parameter is kept at the increased value, otherwise the parameter is decreased by $\alpha_d\%$. The values α_i and α_d are known as the learning rates. One iteration of the algorithm consists of running the simulation and evaluating the error metric for each of the noise parameters. The algorithm continues until the error is less than some convergence value ϵ or until two consecutive iterations are run without producing a new minimum error. This is shown in Algorithm 1, where values of $\alpha_i = 1.5$ and $\alpha_d = 1.1$ were used.

Essentially, the training algorithm trades tuning the actual noise parameters in the Q_k and R_k matrices for tuning the learning rate parameters α_i and α_d.

Algorithm 1 Coordinate Ascent for Discriminative Training

```
while e > e_converge do
  for j = 1 to N_Q + N_R do
    e ← Σ(y_t − h(μ_t))^2
    if e < e_min then
      Q_opt, R_opt ← Q, R
      (Q, R)_{j,j} ← (Q, R)_{j,j} / α_d
    else
      (Q, R)_{j,j} ← (Q, R)_{j,j} * α_i
    end if
  end for
end while
```

Chapter 4

Controls

Control systems are responsible for computing the commands necessary for actuators to cause the trajectory of a system to go from its current state to a desired state, or for answering the question "How do I get there?". There are many different methods that can be used to determine the output commands. One of the more popular and widely implemented control systems is the PID (Proportional, Integral, Differential) controller. Model-based controllers utilize knowledge of the physics of the system to compute appropriate output commands. The robots in these experiments originally used PID controllers for heading control. Later, they were tested with a model-based contoller that takes advantage of Lyapunov stability theory. Both types of controllers described in this chapter take in state estimates from the Kalman filter of Chapter 3 and output linear and angular velocity commands to the robots.

4.1 PID

PID controllers use the current state estimate to determine the errors between the desired state and the current state. The goal of the PID controller is to drive those errors to zero based on a number of criteria, including rise time, settling time, steady state error and overshoot, as shown in Figure 4.1. In that Figure the green line is the ideal output while the blue and purple lines are typical results. The orange line shows what happens when a system becomes unstable. Point m represents the desired state, $m - n$ represents steady state error, point a is the rise

time for the blue line and point b is the settling time for the blue line.

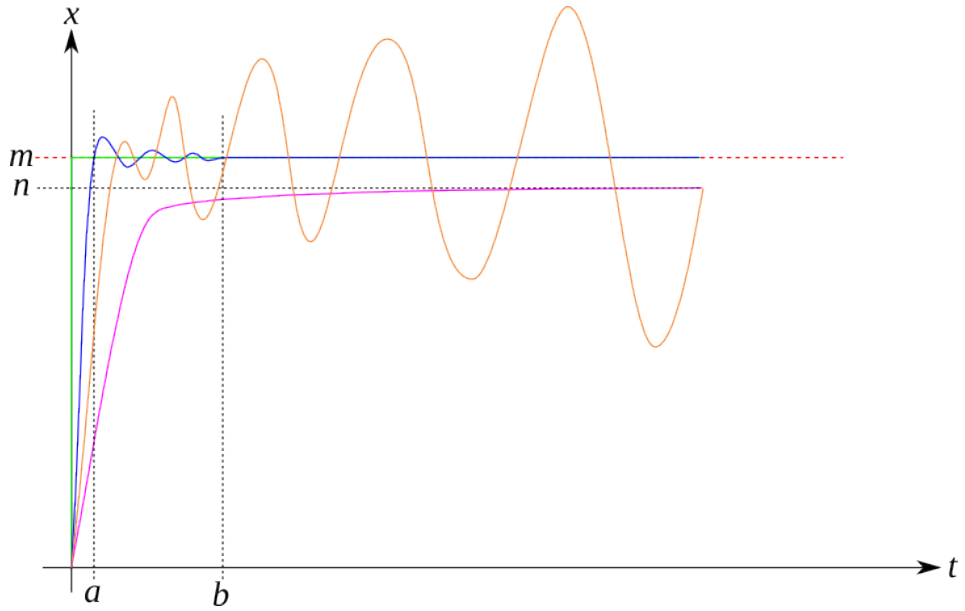


Figure 4.1: PID Controller Responses

The process used to compute an output command using a PID controller uses three separate errors and a gain for each of the distance and heading errors. Looking at the PID controller for heading, the errors are

$$E_P = \psi_{\text{ref}_k} - \psi_k$$
$$E_I = \sum_{i=0}^{k} E_{P_i} * \Delta_T$$
$$E_D = \frac{\psi_k - \psi_{k-1}}{\Delta_T}$$

where ψ_{ref_k} is the desired heading at the current time, ψ_k is the current heading estimate, ψ_{k-1} is the previous heading estimate and Δ_T is the time elapsed since the last PID control calculation was performed. The contribution of each error is then weighted by a gain to obtain the final output command, ω, such that

$$\omega = K_P * E_P + K_I * E_I + K_D * E_D$$

where K_P is the proportional gain, K_I is the integral gain and K_D is the differential gain.

The only parameters available to tune PID controllers for performance are the gains. There are some rules of thumb for tuning gains properly as described in [Ziegler 42] that can work as a good starting point. A basic knowledge of what the effects are when modifying the different gains is summarized in Table 4.1. The difficulty in using PID controllers for the small robots used in these experiments is that the gains must be tuned for specific operating scenarios, so that a set of gains that work well at full speed on asphalt do not work at all when the robot is driving in soft sand at any velocity. PID controllers work best when they only have to reject a small range of disturbances, however, robots are often required to operate in environments with a large range of disturbances.

When the characteristics of a robot are changed the PID gains must also be modified to reflect those changes. These characteristics include mass, center of mass, treads, motors and payloads as these all affect the dynamics of the system. Gain scheduling is the process of tuning a system to use a different set of gains based on the operating environment or characteristics of the robot and is very time consuming. This motivates the search for a better control system for these small robots.

Table 4.1 shows how increasing the different gain values affects the performance of the controller output in terms of rise time, overshoot, settling time and steady state error. It can be seen that a large amount of coupling exists between the gains and attempting to fix one aspect of the controller output can have unintended consequences that cause other aspects of controller output to perform poorly. This is in contrast to the gains that arise for model-based controllers as seen in Section 4.2.6.

Table 4.1: Effects on Controller Output of Modifying PID Gains

Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small Decrease	Decrease	Decrease	None

4.1.1 PID Implementation

The PID controller used by SSCPAC was only for angular rate output while the linear velocity output was set to a maximum allowable velocity and then a simple ramp function was used to slow the robot down as it approached a waypoint. In conjunction with the PID controller, a simple local path planner was developed following work in [Hogg 02]. This path planner uses a carrot placed a reasonable distance in front of the robot on the path from the previous to the current waypoint and the carrot position, rather than the waypoint position, is used to correct the heading. When the waypoint is a large distance from the robots current position, this causes the PID controller to move the robot onto the path faster than it would if the heading error were only based on the angle to the waypoint . The carrot could also be extended along the path after the waypoint so that the robot would not stop at each intermediate waypoint but would instead cut the corner on the way to the next waypoint. One of the difficulties in using the PID controller, and a motivation for finding a different control scheme, is that as the maximum allowable linear velocity was set to larger values the performance of the controller would work poorly when the waypoints were spaced close together as is the case when obstacles are present. The converse is also true. As mentioned previously, gain scheduling is a time intensive task but it is possible that it would have provided a working solution.

4.2 Model Based Controller

The motivation to find a replacement for the PID controller was caused by one major shortcoming. When the PID gains were set to work well when the robot was running at full speed the robot did not work well when slowing down, as is typically the case when obstacles are encountered. Tuning the gains to work well at slow speeds caused the robot to drive poorly at fast speeds. As will be shown in Chapter 5, the model-based controller works well at varying speeds. There are also cases where the PID controller becomes unstable and cannot converge to a goal position. The model-based controller does not suffer from this deficiency.

An alternative control method to using classical PID controllers is to use

model-based controllers based on Lyapunov stability theory. An intuitive way to conceptualize controllers based on this stabilty theory is by thinking of them as decreasing the overall energy of a system, even when the variables involved in the control Lyapunov functions do not represent energy [Khalil 02].

The theorem for Lyapunov stability states that, for an equilibrium point $x = 0$ and a domain $D \subset \mathbb{R}^n$ that contains $x = 0$, and for which there is a function $V : D \to \mathbb{R}$ that is continuously differentiable and has the following properties

$$\begin{aligned} V(0) &= 0 \\ V(x) &> 0 \in D - \{0\} \\ \dot{V}(x) &\leq 0 \in D - \{0\} \end{aligned} \tag{4.1}$$

then the equilibrium point $x = 0$ is stable. Additionally, if

$$\dot{V}(x) < 0 \in D - \{0\} \tag{4.2}$$

then $x = 0$ is asymptotically stable.

It is not always possible to find such a control Lyapunov function V, but when one is found then this theorem holds and the "energy" of the system will always be positive and decreasing. The "energy" can consist of any variables that make $V(0) = 0$ and $V(x) > 0$ and consist of a combination of the errors that are to be minimized in a system. In that case the errors are always decreasing since $\dot{V}(x) < 0$ and the system will reach the desired state.

Two different modes of the model-based controller were developed and tested during these experiments. The difference between the two modes is the behavior of the robot as it approaches intermediate waypoints while navigating to its final destination. The first mode is considered to be useful for a parking behavior where the robot stops at each waypoint, including the intermediate waypoints. The second mode is a slightly modified version of the parking behavior, where the distance error is extended beyond each intermediate waypoint so that the robot will drive through the waypoints without stopping (although the robot often does slow down some) until it reaches its final destination. This second mode will be referred to as the driving mode. During the following development of the control law the parking

mode will be described fully and then extensions will build directly on top of the parking mode to describe the driving mode in Section 4.2.5.

4.2.1 Error Terms for the Model-Based Controller

Several groups have implemented model-based controllers rooted in Lyapunov stability theory in simulation including [Micaelli 93], [Aicardi 94], [Aicardi 95], [Rusu 05], [Gulati 08]. Only recently have these results been applied to actual robots [Kim 05], [Lapierre 06], [Lapierre 07], [Nüchter 07]. All of these groups describe a method for constructing a control Lyapunov function based on the kinematics of a differential drive robot, similar in nature to the small robots considered in these experiments and typically referred to as unicycle-like robots in the literature.

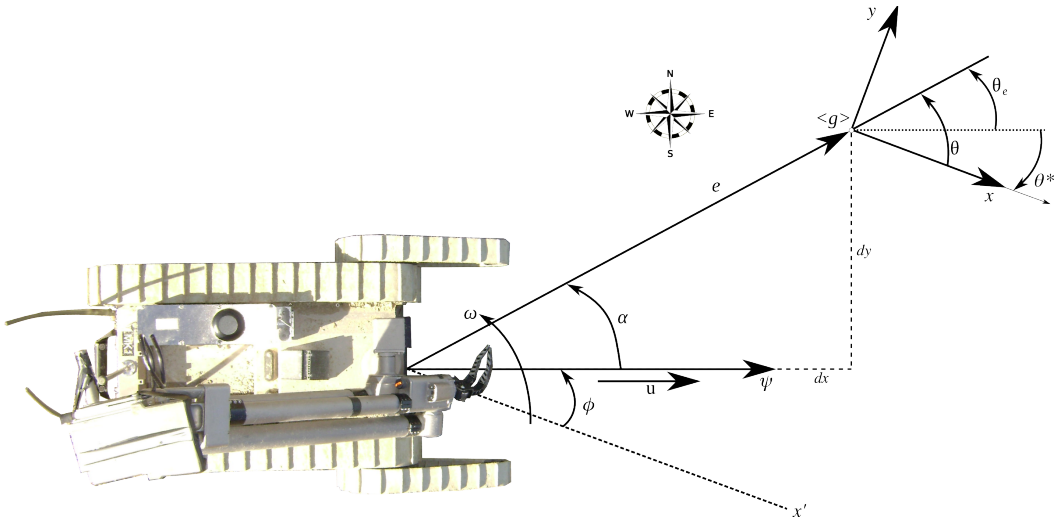


Figure 4.2: Packbot Coordinate System for Model Based Controller

Figure 4.2 will be used as the basis for the equations that follow and will be explained in further detail here. Since life is never easy, this controller has been developed with the global x-axis pointing East, the local x-axis pointing to the right of the robot while the global y-axis is North and the local y-axis is pointing forward of the robot. Since the Kalman filter has the local x-axis forward and the local y-axis pointing left the yaw state has to be transformed into the new coordinate system.

Given a robot at some arbitrary initial position we want it to move to a

goal position with a specific heading, where the goal position is the origin of the $< g >$ coordinate system and the desired heading is along the x-axis of the $< g >$ coordinate system, labeled $\theta^{\star}$. The states that define the robot's current position, heading, and linear and angular velocities are calculated using the Kalman filter of Section 3.2.5 whereas the goal position and heading are given as inputs to the robot. Note that the linear velocity is given by u and the angular velocity by ω.

The control system attempts to force three separate errors to zero:

- e, the magnitude of the translation error vector as measured between the current position and goal position,
- θ, the angle between the desired heading and the translational error vector e,
- α, the angle between the current heading and the translational error vector e.

The reason that those three particular errors were selected is driven by the fact that the robots are nonholonomic (they are only able to move forwards or backwards along the direction of their current heading and cannot move sideways). For holonomic robots a reasonable controller that gets the robot to a desired position with a desired heading would simply move at an angle towards the goal position and along the way correct its heading. Since that is not possible for the robots here some tradeoffs are required and those come in the form of the two angle errors, θ and α. One way that a controller for nonholonomic vehicles could work is to rotate until the robot is pointed towards the goal position, then move in a straight line to the goal position, stop and finally rotate in place again to the desired heading. However, that is boring and not very efficient as it would mean that the robot would have to stop at each waypoint along a route so that it could correct its heading to the next waypoint. The approach taken with the model-based controller described here is to let the robot have the ability to align itself with the goal heading by zeroing out θ and α before it arrives at the goal position. This approach will allow the robot to maintain speed when it arrives at intermediate waypoints along the route.

Additional variables used to calculate the errors, shown in Figure 4.2, include:

- $\theta^{\star}$, the target heading in the global coordinate system which is aligned with the x-axis of $< g >$,

- θ_e, the angle of the error vector in the global coordinate system,
- ψ, the current heading of the robot in the global coordinate system,
- ϕ, the difference between the target heading $\theta^\star$ and the current heading ψ in the global coordinate system.

4.2.2 Unicycle-like Robot Kinematics

A kinematic model of the robot is used to predict how the robot moves without considering the effects of the mass of the robot or outside forces (such as friction between the robot tracks and the ground) with

$$\begin{aligned} \dot{x} &= u\cos\phi \\ \dot{y} &= u\sin\phi \\ \dot{\phi} &= \omega. \end{aligned} \tag{4.3}$$

The expressions in (4.3) are then converted to a polar coordinate representation via a coordinate transformation. This gives expressions for the errors that the control system is attempting to drive to zero such that

$$\begin{aligned} e &= \sqrt{x^2 + y^2} \\ \theta &= \text{atanh}(y, x) \\ \alpha &= \theta - \phi. \end{aligned} \tag{4.4}$$

Combining (4.4) with (4.3) results in

$$\begin{aligned} \dot{e} &= -u * \cos(\theta - \phi) \\ \dot{\theta} &= u\frac{\sin\alpha}{e} \\ \dot{\phi} &= \omega. \end{aligned}$$

Finally, replacing α with $\theta - \phi$ results in a kinematic model of

$$\begin{aligned} \dot{e} &= -u\cos\alpha \\ \dot{\alpha} &= -\omega + u\frac{\sin\alpha}{e} \\ \dot{\theta} &= u\frac{\sin\alpha}{e}. \end{aligned} \tag{4.5}$$

4.2.3 Control Lyapunov Function

The control Lyapunov function is selected to be positive, contain all three error states and separate the distance error from the angle errors. It is given by

$$V = V_1 + V_2 = \frac{1}{2}\lambda e^2 + \frac{1}{2}\left(\alpha^2 + h\theta^2\right) \tag{4.6}$$

where λ and h are positive constants that can be used to tune the controller output (see Section 4.2.6). Using the kinematics equations in (4.5), the derivative of each term of the candidate control Lyapunov function can be found as

$$\begin{aligned} \dot{V}_1 &= \lambda e\dot{e} = \lambda e(-u\cos\alpha) = -\lambda e u\cos\alpha \\ \dot{V}_2 &= \alpha\dot{\alpha} + h\theta\dot{\theta} \\ &= -\alpha\omega + \alpha u\frac{\sin\alpha}{e} + h\theta u\frac{\sin\alpha}{e} \\ &= \alpha\left(-\omega + u\frac{\sin\alpha}{e} + h\theta u\frac{1}{\alpha}\frac{\sin\alpha}{e}\right) \\ &= \alpha\left(-\omega + u\frac{\sin\alpha}{\alpha}\frac{(\alpha + h\theta)}{e}\right) \end{aligned} \tag{4.7}$$

and from (4.7) the total derivative is found as

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\lambda e u\cos\alpha + \alpha\left(-\omega + u\frac{\sin\alpha}{\alpha}\frac{(\alpha + h\theta)}{e}\right). \tag{4.8}$$

Now, it needs to be shown that $\dot{V} \leq 0$ which can be done by showing that $\dot{V}_1 \leq 0$ and $\dot{V}_2 \leq 0$. This is true for $\dot{V}_1$ if u takes the form

$$u = \gamma e\cos\alpha \tag{4.9}$$

where γ is a positive constant different from λ. Substituting this value of u into (4.7) results in

$$\dot{V}_1 = -\lambda e u \cos\alpha = -\lambda\gamma e^2 \cos^2\alpha \leq 0. \tag{4.10}$$

Since $V_1 > 0$ and $\dot{V}_1 \leq 0$ we have that V_1 converges to a positive defined limit.

Replacing u from (4.9) in the expression for $\dot{V}_2$ in (4.7) results in

$$\begin{aligned}\dot{V}_2 &= \alpha\left(-\omega + u\frac{\sin\alpha}{\alpha}\frac{(\alpha+h\theta)}{e}\right)\\ &= \alpha\left(-\omega + \gamma e\frac{\cos\alpha\sin\alpha}{\alpha}\frac{(\alpha+h\theta)}{e}\right)\\ &= \alpha\left(-\omega + \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha+h\theta)\right).\end{aligned} \tag{4.11}$$

Similar to the analysis of $\dot{V}_1$, $\dot{V}_2$ is negative definite if ω takes the form

$$\omega = k\alpha + \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha+h\theta) \tag{4.12}$$

where k is a positive constant. Substituting this value of ω into (4.11) gives

$$\begin{aligned}\dot{V}_2 &= \alpha\left(-k\alpha - \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha+h\theta) + \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha+h\theta)\right)\\ &= -k\alpha^2 < 0\end{aligned} \tag{4.13}$$

and V_2 converges asymptotically to a positive defined limit.

Substituting $\dot{V}_1$ from (4.10) and $\dot{V}_2$ from (4.13) into the expression for $\dot{V}$ (4.8) yields

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\lambda\gamma e^2\cos^2\alpha - k\alpha^2 \leq 0.$$

This expression is equal to zero if and only if $\alpha = 0$ *and* $e = 0$, which only occurs when the robot has reached its goal pose. Additionally, it can be shown that $\theta = 0$ when $\dot{V} = 0$ by invoking Barbalat's lemma [Aicardi 95]. Therefore, at any point along the robots trajectory, other than the goal position and heading, the function $\dot{V}$ is negative definite and satisfies the properties for asymptotic stability given by

(4.1) and (4.2).

Using the expressions for u in (4.9) and ω in (4.12) and substituting those values back into the kinematic model from (4.5) gives

$$\begin{aligned}
\dot{e} &= -u\cos\alpha = -\gamma e\cos^2\alpha \\
\dot{\alpha} &= -\omega + u\frac{\sin\alpha}{e} \\
&= -(k\alpha + \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha + h\theta)) + \gamma e\cos\alpha\frac{\sin\alpha}{e} \\
&= -k\alpha - \gamma\cos\alpha\sin\alpha + \gamma\cos\alpha\sin\alpha - \gamma h\theta\frac{\cos\alpha\sin\alpha}{\alpha} \\
&= -\left(k\alpha + \gamma h\theta\frac{\cos\alpha\sin\alpha}{\alpha}\right) \\
\dot{\theta} &= u\frac{\sin\alpha}{e} = \gamma e\cos\alpha\frac{\sin\alpha}{e} = \gamma\cos\alpha\sin\alpha
\end{aligned} \tag{4.14}$$

Combining (4.9) and (4.12) gives the following control law to replace the PID controller from Section 4.1:

$$\begin{aligned}
u &= \gamma e\cos\alpha \\
\omega &= k\alpha + \gamma\frac{\cos\alpha\sin\alpha}{\alpha}(\alpha + h\theta)
\end{aligned} \tag{4.15}$$

4.2.4 Calculating Control Law Variables

The method used to calculate the variables needed for the control law in (4.15) based on Figure 4.2 is:

1. dx_e is the horizontal component of the error vector e.
2. dy_e is the vertical component of the error vector e.
3. Use $e = \sqrt{dx_e^2 + dy_e^2}$ to get the distance to the waypoint on the current path segment.
4. ψ is the current heading of the robot in the world coordinate frame and is calculated in the Kalman filter based on the results from Chapter 3. Note that this has to be rotated so that it works in the current coordinate system.
5. θ_e is the angle of the error vector e in the global coordinate syste in the global coordinate system and is found as $\theta_e = \text{atan2}(dy_e, dx_e)$.

6. $\theta^\star$ is the desired heading in the global coordinate system and there are multiple ways that it *could* be set. Suppose that the robot is between waypoint 1 and waypoint 2 on a route and that waypoint 3 exists. Let the waypoints have coordinates given by (x_1, y_1), (x_2, y_2) and (x_3, y_3). Then,

   - $\theta^\star = 0$ sets the desired heading as East and $\theta^\star = \frac{\pi}{2}$ sets it as North.
   - $\theta^\star = \psi$ would make the desired heading be the same as whatever the current heading happens to be.
   - $\theta^\star$ can be sent in as an additional parameter of a waypoint, say from MOCU via JAUS.
   - $\theta^\star$ can be the angle from the previous waypoint to the current waypoint. This is found by using $dx_{\theta^\star} = x_2 - x_1$, $dy_{\theta^\star} = y_2 - y_1$ and $\theta^\star = \text{atan2}(dy_{\theta^\star}, dx_{\theta^\star})$.
   - $\theta^\star$ can be the angle from the current waypoint to the next waypoint. This is found by using $dx_{\theta^\star} = x_3 - x_2$, $dy_{\theta^\star} = y_3 - y_2$ and $\theta^\star = \text{atan2}(dy_{\theta^\star}, dx_{\theta^\star})$.

7. $\phi = \theta^\star - \psi$.

8. $\theta = \theta^\star - \theta_e$.

9. $\alpha = \theta - \phi$.

4.2.5 Model Based Driving Mode

The parking mode described above can be extended such that the control law will cause the robot to drive through intermediate waypoints on a route rather than stop at each waypoint. The basic idea is that the target position as given in the goal frame $< g >$ of Figure 4.2 is moved such that the distance error e is increased. Since the linear velocity output of the controller is a function of e (c.f. (4.15)), that output will increase causing the robot to drive faster through the intermediate waypoints. As described in [Aicardi 95], an effective means of moving

the goal frame is to control its rate of motion, $\dot{s}$, via the function

$$\dot{s} = \begin{cases} 0, & V = \lambda e^2 + (\alpha^2 + h\theta^2) > \epsilon \\ f(e, \alpha, \theta), & V = \lambda e^2 + (\alpha^2 + h\theta^2) \leq \epsilon \end{cases} \tag{4.16}$$

where $0 < \epsilon < \frac{\pi^2}{4}$. The function V describes an ellipsoid such that if any of the errors are larger than the threshold given by ϵ then the goal frame will not be moved and the parking behavior will be used. When the robot is aligned with both the waypoint and the desired heading, causing $(\alpha, \theta) = (0, 0)$, then a reasonable course of action is to maintain the maximum allowable linear velocity. It is for this reason that the design variable λ is typically chosen to be very small to limit the amount of influence that the distance error has on the motion of the goal frame. The implemented function to move the goal frame according to (4.16) was selected to be

$$f(e, \alpha, \theta) = u_{\max} * \max\left(0, 1 - \frac{V}{\epsilon}\right). \tag{4.17}$$

It can be seen from (4.16) and (4.17) that setting ϵ to a small value makes it more difficult for the system to enter the ellipsoidal domain where $V \leq \epsilon$. This also has the effect of decreasing the amount of motion for the goal frame. Note that setting $\epsilon = 0$ results in the parking mode since $\dot{s} = 0$ and the goal frame never moves.

The resulting behavior of the robot has a very natural appearance as this is how humans typically drive. When the road ahead is straight and we want to continue driving straight we keep our eyes focused farther down the road and maintain speed. However, when the road ahead is twisty or we want to make a turn we slow down and focus our eyes closer to our current position.

4.2.6 Practical Considerations for Selecting Gains

The gains h, k and γ in the control law (4.15) determine the resulting trajectory that the robot will use when driving from its current position to the goal point and heading. The first thing to notice is that the linear velocity is at a maximum when the angle error $\alpha = 0$, since that leads to $\cos(\alpha) = 1$ and $u_{\max} = \gamma e$.

To limit the maximum linear velocity of the robot the gain $\gamma = \frac{u_{\max}}{e}$ can be used when $u > u_{\max}$. Typically, in the environments that this testing was performed in, a long path segment was approximately $10m$. Since $2.5m/s$ is a reasonable upper limit for the Packbot linear velocity, a good starting gain is $\gamma = \frac{2.5}{10} = 0.25$.

There are many ways to select what values the gains h and k should take, especially once γ is set. Some of the properties of the kinematics equations (4.14) as the angle error α approaches 0 are helpful in making those selections. When the robot is heading towards the direction of the target position the system involving the errors to be minimized is approximately linear. The key to this linearization is to notice that when α is near zero several terms in (4.14) cancel out since $\alpha \to 0 \Rightarrow \alpha = \cos(\alpha) * \sin(\alpha) = \sin(\alpha)$, and $\cos^2(\alpha) = 1$, as seen in Figure 4.3. When that is the case the following linear approximation holds:

$$\begin{aligned} \begin{bmatrix} \dot{\alpha} \\ \dot{\theta} \end{bmatrix} &= \underbrace{\begin{bmatrix} -k & -\gamma h \\ \gamma & 0 \end{bmatrix}}_{A} \begin{bmatrix} \alpha \\ \theta \end{bmatrix} \\ \dot{e} &= -\gamma e. \end{aligned} \tag{4.18}$$

The output equations (4.15) are also linearized such that

$$\begin{aligned} u &= \gamma e \\ \omega &= (k+\gamma)\alpha + \gamma h \theta. \end{aligned} \tag{4.19}$$

This shows that, in the neighborhood of $\alpha = 0$, the distance error e converges linearly at a rate of $-\gamma$. The angle errors converge at a rate of $-\sigma$, where $-\sigma$ is the real part of the dominant pole of the linear system A in (4.18). The poles of the system are the eigenvalues of A. The reason that the errors converge at those rates is that, for the distance error, $e = \exp(-\gamma t)$ is a solution to the equation $\dot{e} = -\gamma e$ since

$$\tfrac{d}{dt} e = \tfrac{d}{dt} \exp(-\gamma t) = -\gamma \exp(-\gamma t) = -\gamma e.$$

Similarly, the behavior of the angle errors is related to the matrix A. Nonsingular matrices can be factored into the form $A = S \Lambda S^{-1}$, where S contains the eigenvec-

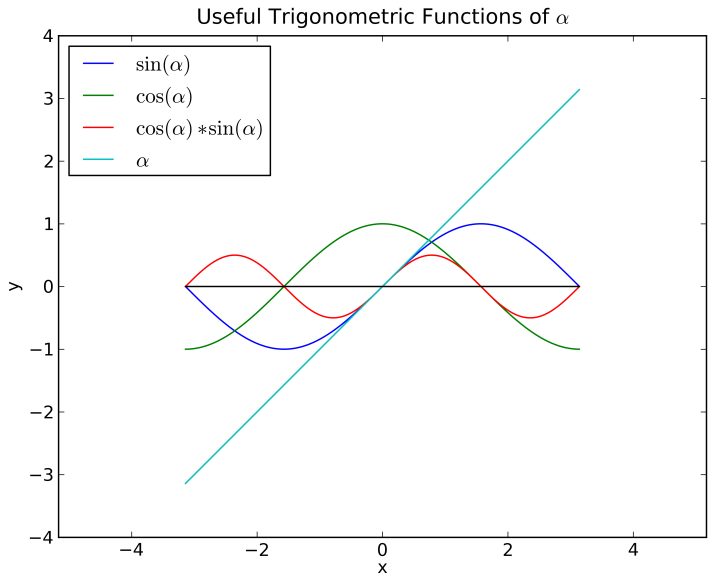


Figure 4.3: Useful Trigonometric Functions of α

tors of A and Λ is a diagonal matrix containing the eigenvalues of A. The solution to the differential equation is

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\theta} \end{bmatrix} = \exp(At) = \exp(S\Lambda S^{-1}t) = S\exp(\Lambda t)S^{-1}$$

and $\dot{\alpha}$ has the solution $\alpha = \exp(\lambda_\alpha t)$, while $\dot{\theta}$ has the solution $\theta = \exp(\lambda_\theta t)$.

In parking mode three reasonable design considerations in selecting the gains are:

- limit the maximum linear velocity,
- find critically damped gains,
- have angle errors converge before distance error.

Limiting the maximum linear velocity has been discussed already and involves the selection of γ. The damping of the system A is determined by

$$\zeta = \frac{\lambda_\alpha(A)}{\lambda_\theta(A)}$$

and the gains are critically damped when $\zeta = 1 \Rightarrow \lambda_\alpha(A) = \lambda_\theta(A) = \lambda$. This causes both angle errors, α and θ, to converge to zero at the same rate. When the system is critically damped the angle errors will converge faster than the distance error if $\sigma > \gamma$, since that means the rate of convergence for the angle errors is greater than the rate of convergence for the distance error. In the case where the A matrix is overdamped $\zeta > 1 \Rightarrow \lambda_\alpha(A) > \lambda_\theta(A)$ then α will converge to zero faster than θ. Conversely, when A is underdamped $\zeta < 1$ and θ converges faster than α.

By selecting the gains to satisfy the design considerations above, the robot will align itself with the goal heading before arriving at the goal point. The result is that the robot will smoothly decrease its linear velocity as it approaches the goal point. One alternative occurs when the distance error converges faster than the angle errors, $\gamma > \sigma$, and the robot will get to the goal point and then finish aligning with the goal heading. In parking mode this leads to several problems: (i) the robot does not look as intelligent since the trajectory does not appear to be as "natural" as the way a human would drive and (ii) rotating in place is typically a much more difficult maneuver for tracked robots to perform and increases fatigue on the treads compared to turning while driving forward. In driving mode, the case when $\gamma > \sigma$ is not such a problem because the linear velocity is kept large enough that the robot does not slow down as often.

To deterministically discover gains that will satisfy the properties discussed for parking mode, two conditions must be met:

- $\zeta = 1$
- $\sigma > \gamma$

For $\zeta = 1$, it is necessary to first determine the eigenvalues of A.

$$\det(A - \lambda I) = \begin{vmatrix} -k - \lambda & -h\gamma \\ \gamma & -\lambda \end{vmatrix} = 0$$

$$\Rightarrow (-k - \lambda)(-\lambda) + h\gamma^2 = 0$$

$$\Rightarrow \lambda^2 + k\lambda + h\gamma^2 = 0$$

$$\Rightarrow \lambda = \frac{-k \pm \sqrt{k^2 - 4h\gamma^2}}{2}$$

For $\lambda_\alpha(A) = \lambda_\theta(A)$, the term inside the square root must be equal to zero resulting in

$$\begin{aligned} k^2 - 4h\gamma^2 &= 0 \\ \Rightarrow k &= \sqrt{4h\gamma^2} = 2\gamma\sqrt{h}. \end{aligned}$$

To find $\sigma > \gamma$ constrained by $k = 2\gamma\sqrt{h}$ it is necessary to have

$$\begin{aligned} \sigma = -\lambda = \tfrac{k}{2} &> \gamma \\ \Rightarrow k &> 2\gamma \end{aligned}$$

The two equations can be combined to find h such that

$$\begin{aligned} k = 2\gamma\sqrt{h} &> 2\gamma \\ \Rightarrow h &> 1 \end{aligned}$$

Using this method, any two gains can be selected and the third gain can be found which satisfies the above properties. For example, setting $\gamma = 0.25$ to limit the maximum linear velocity and $h = 1.1$ to keep it small but greater than one leads to $k = 2\gamma\sqrt{h} = 0.52$. From this it can be seen that $\lambda_\alpha(A) = \lambda_\theta(A) = 0.26 \Rightarrow \zeta = 1$ so the system A is critically damped and $\sigma = 0.26 > 0.25 = \gamma$ so that the angle errors will converge faster than the distance error.

When using (4.16) to move the goal frame and put the robot into driving mode, it can be more desirable to generate trajectories such that the robot stays on a straight line path to the next waypoint. The result is that the angle errors are decreased and the robot drives faster as it goes through intermediate waypoints. One way to have the robot drive towards the waypoint is to set the gains such that the distance error converges faster than the angle errors and have α converge faster than θ. An example of the gains that produce these effects is $\gamma = 0.25$, $h = 0.33$ and $k = 0.30$. Another way to cause the robot to drive on the line segment connecting the previous and current waypoints is to set the target heading to be the angle from the previous waypoint to the current waypoint.

The model-based controller developed in this chapter, along with the analysis

of selecting appropriate gains, replaces the original PID controller when answering the question "How do I get there?".

Chapter 5

Results

The Kalman filter and controls algorithms discussed in this thesis were implemented in software on a Packbot. They were then run in an open field with an uneven surface consisting of dirt, gravel and asphalt, as seen in Figure 5.1. This testing area is difficult for the robots to navigate because pitch, roll and elevation have significant changes, plus the loose dirt and gravel can cause the tracks to slip leading to erroneous encoder data. The environment, in addition to routes that force the robot to change linear and angular velocities, excites all the modes of the system model. Some effects, such as track slip, are not modeled and have an effect on the algorithms. All of these conditions rigorously test the estimation and control algorithms.

5.1 Kalman Filter Results

In Chapter 3, several aspects of the Kalman filter were modified with the goal of improving robot state estimation. Two metrics have been used to quantify the difference in performance of the Kalman filter after making changes to the algorithm. The first metric is the root mean square (RMS) error over the entire course. The second metric is the amount of error in the position estimate before and after the robot runs a route, where the percentage is the position error divided by the total amount of distance traveled during the route. Also included is the total distance traveled during the run that the data was logged. The different changes that were tested were the baseline case, fixing the Kalman filter implementation

Figure 5.1: Robot Test Area

bugs described in Section 3.3, hand tuning the Kalman filter noise models and, finally, using noise models learned from the training algorithm from Section 3.5.1. The results for Kalman filter performance are shown in Table 5.1.

Table 5.1: Kalman Filter Performance

Stage	RMS Error (m)	Return Error (%)	Distance Traveled (m)
Baseline	10.62	1.51	224.22
Fix Bugs	4.38	0.79	182.70
Hand Tuned	4.87	1.32	184.09
Trained	3.64	0.99	332.62

The most significant improvement came from fixing the bugs in the Kalman filter code. Using the training algorithm to tune the parameters in the noise models showed a marked improvement over the hand tuned case and was also better on average than the case where the bugs were fixed. Although the trained noise models case has a slightly greater error in the difference between starting and stopping positions that run was much greater in total distance traveled and a return error of 0.99% over a $332m$ run is more than sufficient to perform retrotraverse in most environments where the robots are used.

5.1.1 Kalman Filter Effects on Controller

Early on during testing, before the bugs were discovered and fixed in the Kalman filter, several runs were logged that show how estimation errors can affect the model-based and PID controllers. In this Section the data was collected before the bugs in the Kalman filter were fixed (Section 3.3), but after training of the noise models was used (Section 3.5.1). The results are interesting because they show that the previous performance of the PID controller could break down and become unusable. It also shows that the training algorithm can improve a filter with equations that are not optimal. The results show that not only is the Kalman filter performance better after using the training algorithm, but it can also be seen that the model-based controller has much better performance than the PID controller when there are simulated obstacles along the route.

In Figures 5.2 - 5.5, the same route was driven and the waypoints making up the route are shown on the overhead images. The red line is the Kalman filter position esimate, the blue line is raw GPS measurements and the green line is DPGS measurements.

The results of using the model-based controller with original noise models is shown in Figure 5.2. It can be seen that going from waypoint 4 to waypoint 5 the Kalman filter started to diverge significantly from the GPS measurements. The filter estimate then snaps back to line up with the GPS measurements. This is done outside the standard Kalman filter equations and is used to bound the position error.

Running the model-based controller with learned noise models resulted in the positions shown in Figure 5.3. Here it can be seen that the Kalman filter position estimate is much closer to the GPS measurements for the entire route and that in certain places, such as near waypoint 4 and between waypoint 6 and waypoint 7, the Kalman filter is able to remove discontinuities in the GPS position. Also, in the section of the route with waypoints close to each other to simulate navigation around obstacles, the model-based controller is able to drive through the waypoints.

The PID controller with original noise models is shown in Figure 5.4. Similar results are seen in regards to the Kalman filter position estimate snapping to the GPS measurements when significant divergence occurs between wayoint 2 and

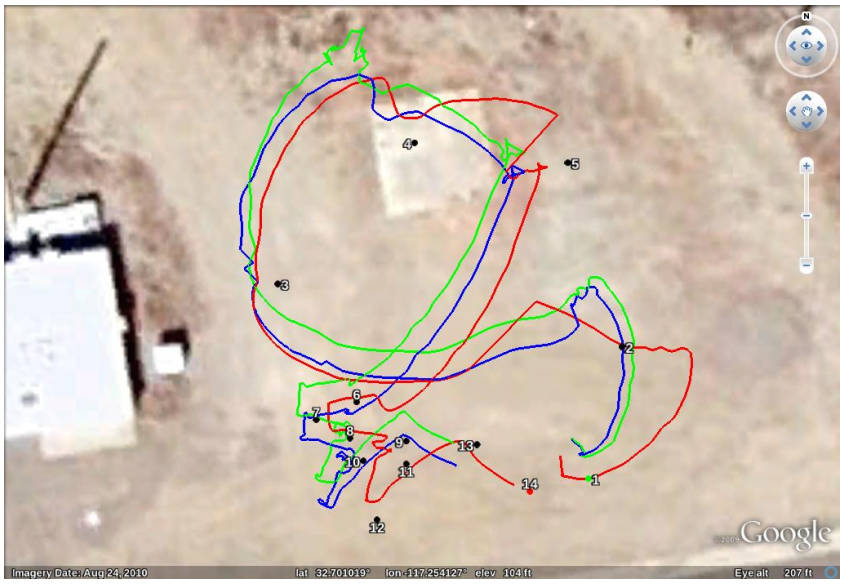


Figure 5.2: Model Based Controller with Original Noise Models

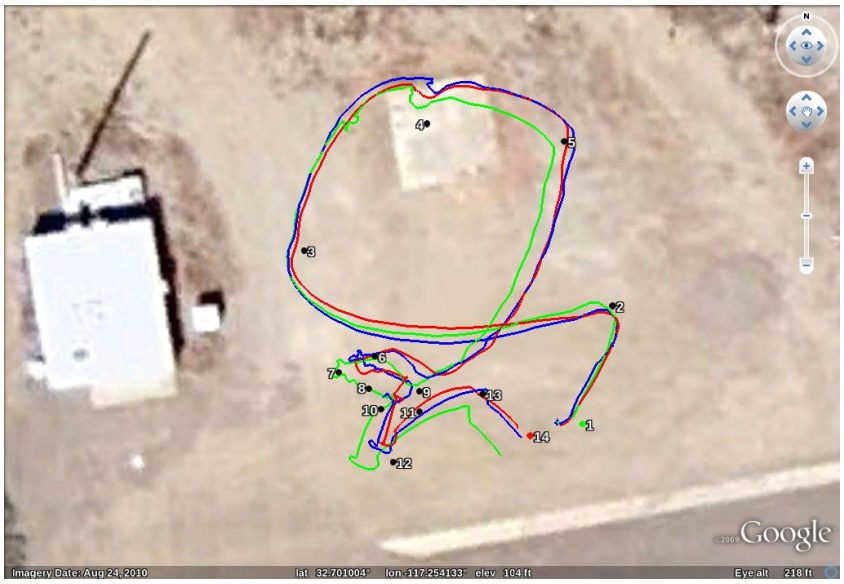


Figure 5.3: Model Based Controller with Learned Noise Models

waypoint 3. The PID controller has trouble in the section of the route between waypoint 6 and waypoint 12, the area where the waypoints were set up close together to simulate obstacles. In this section of the route, the Kalman filter yaw estimate becomes very poor as the robot was driving in circles trying to reach the waypoints. As a result, the Kalman filter position estimate diverges and would have snapped back to the GPS measurement if the route were not finished at waypoint

14.

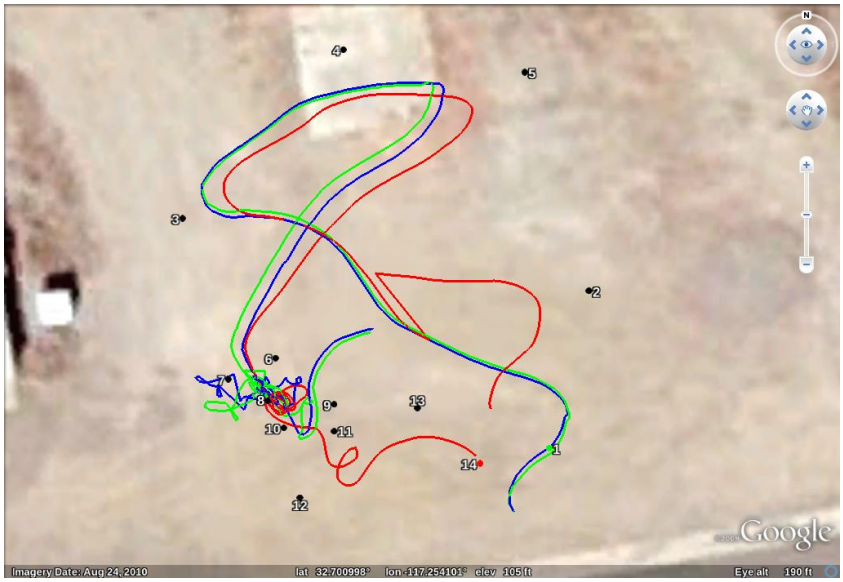


Figure 5.4: PID Controller with Original Noise Models

The PID controller with learned noise models has similar trouble navigating the waypoints that are close together, as seen in Figure 5.5. However, with the learned noise models, the Kalman filter position estimate stays much closer to the GPS measurements even when the robot is driving in circles. The improved position estimate is also indicative of an improved yaw estimate from the Kalman filter.

5.2 Model Based Controller Results

The linear and angular velocity outputs that were the result of running a simple route with the PID controller are shown in Figure 5.6. This compares to the outputs of the model-based controller shown in Figure 5.7, where it can be seen that the angular velocity is smooth and the linear velocity drives much faster, using 100% effort at times, but smoothly decelerating as waypoints are reached. The linear velocity output does not go to zero until the final waypoint is reached as a consequence of the moving target frame described in Section 4.2.5. The errors for the model-based controller are shown in Figure 5.8, showing that e, α and θ decrease linearly at the rate given by the gain γ and the eigenvalues of the matrix

Figure 5.5: PID Controller with Learned Noise Models

A, composed of the gains h and k.

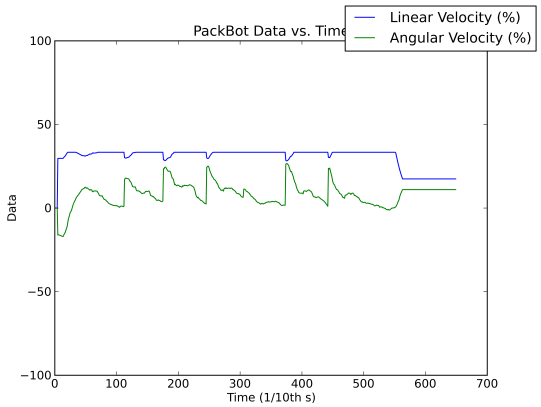


Figure 5.6: Typical PID Controller Velocity Output

5.2.1 Controller Comparison

For both PID (Section 4.1, Table 4.1) and the model-based controller (Section 4.2.6) gains have to be selected. The difference in selecting gains is that stability is the goal when selecting PID gains whereas stability is guaranteed with the model-based controller. More advanced behaviors, such as three point turns, are a consequence of the control law in (4.15) derived using a kinematic model of the

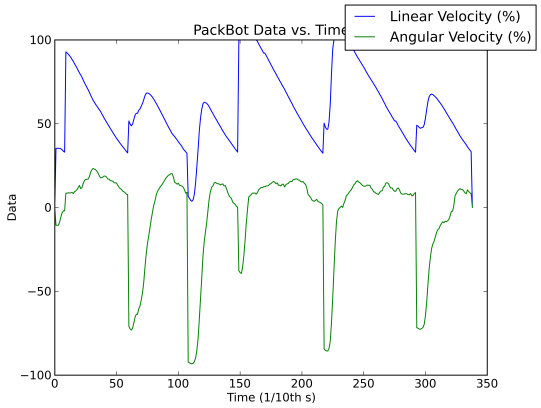


Figure 5.7: Typical Model Based Controller Velocity Output

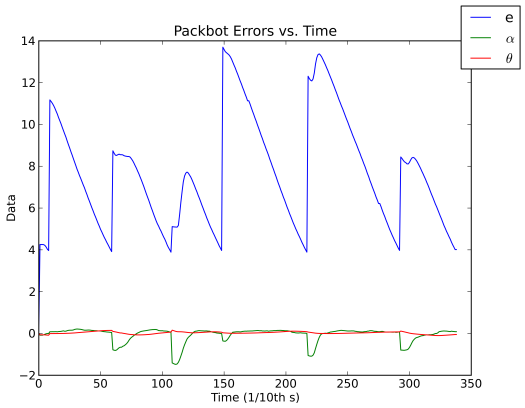


Figure 5.8: Model Based Controller Errors

robot. The other very large difference between the controllers is that a table of gains must be tuned for the PID controller to work at varying linear velocities, but the model-based controller works very well at different linear and angular velocities. Also, when properties of the robot, such as mass, are changed by using different payloads for the system the entire table of PID gains must be retuned while, at most, one set of gains must be retuned for the model-based controller. This last benefit of the model-based controller makes it very appealing to use in fielded systems, since it reduces the amount of maintenance and effort required to keep the robot in optimal working condition. The model-based controller will also allow for improved navigation performance near obstacles, which leads directly to improved autonomy for the robot. The local path planner is much simpler with the model-based controller, since it simply requires moving the target frame based on

the calculated errors. The PID controller uses an entirely new algorithm that is not a normal part of the control algorithm to set the carrot following mode [Hogg 02].

Many different controller configurations are possible based on varying the gains used and the calculation of the desired heading at waypoints. The tested configurations are shown in Table 5.2.

Table 5.2: Controller Setups

Setup	Controller	γ	h	k	$\theta^\star$
1	PID	N/A	N/A	N/A	N/A
2	Model Based	0.25	0.33	0.30	CN
3	Model Based	0.25	0.33	0.30	PC
4	Model Based	0.25	1.10	$2\gamma\sqrt{h}$	CN
5	Model Based	0.25	1.10	$2\gamma\sqrt{h}$	PC

Note that, in all cases for the model-based controller, the desired goal heading $\theta^\star$ for the final waypoint was set to be the angle from the previous waypoint to the last waypoint.

Each controller configuration was run on several courses that test specific behaviors.

- Allowing for maximum velocities to be sustained, with large amounts of open space on the route, by having long path segments shown in Figure 5.9. This route used $\lambda = 0.001$, $\epsilon = 2.0$, $R_{\text{cap}} = 1.0$. There are 13 waypoints covering a total of $90.73m$ with an average distance between them of $7.56m$.

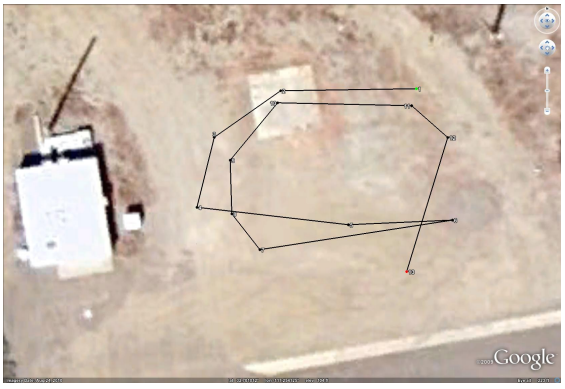


Figure 5.9: Route with Long Path Segments

- Short path segments between waypoints to simulate obstacles along the route shown in Figure 5.10. This route used $\lambda = 0.001$, $\epsilon = 1.0$, $R_{\text{cap}} = 0.2$. There are 27 waypoints covering a total of $43.18m$ with an average distance between them of $1.66m$.

Figure 5.10: Route with Short Path Segments

- A mixture of long and short path segments shown in Figure 5.11. This route used $\lambda = 0.001$, $\epsilon = 1.5$, $R_{\text{cap}} = 0.5$. There are 13 waypoints covering a total of $70.87m$ with an average distance between them of $5.45m$.

Figure 5.11: Route with Mixed Length Path Segments

The variable R_{cap} is the capture radius for each waypoint, and is used to determine when the robot is close enough to the current waypoint, so that the next waypoint should be set as the current target.

The results using the setups described are shown for the different routes in Tables 5.3 - 5.5, where the metrics used to compare the controller performance are

- the time required to complete the course,
- the average cross track (XT) error during the course. This is to show whether the robot followed a straight line between waypoints, which is important when obstacles are in the area, and not important when large amounts of open space are available for navigation. One issue is that some operators become nervous when the robot trajectory has a large curvature and deviates from the straight line path even if open space is available,
- the maximum velocity achieved by the robot during the course,
- the number of times that the robot stops, where a stop is defined as the robot having a velocity that drops below a threshold. The threshold used was 10% of the maximum velocity.

Table 5.3: Controller Comparison on Open Space Route

Setup	Time (s)	XT (m)	$u_{\max}$ (%)	Stops
1	82.67	0.61	33.3	1
2	75.12	2.52	100.0	3
3	71.42	2.71	100.0	4
4	110.09	2.42	100.0	14
5	75.45	2.80	100.0	5

Table 5.4: Controller Comparison on Simulated Obstacle Route

Setup	Time (s)	XT (m)	$u_{\max}$ (%)	Stops
1	N/A	N/A	N/A	N/A
2	134.92	0.85	42.52	20
3	75.92	0.94	50.86	21
4	232.09	0.79	27.90	31
5	75.99	0.91	49.96	23

The PID controller only worked well on the route with long path segments. For the short and mixed path segment routes, it actually became sad to watch and the tests were aborted because the robot was just spinning in circles and running the batteries down, similar to what was seen in Figure 5.5.

Table 5.5: Controller Comparison on Mixed Route

Setup	Time (s)	XT (m)	$u_{\max}$ (%)	Stops
1	N/A	N/A	N/A	N/A
2	103.39	2.51	100.0	11
3	76.91	2.98	100.0	12
4	164.65	2.34	100.0	22
5	83.84	2.87	100.0	12

All of the model-based controllers are fairly similar. However, it can be seen that Setup 2 consistently has longer run times but smaller cross track errors. This indicates that during those runs, the robot was on the line between the previous and current waypoints with a heading towards the waypoint, but the robot slowed down at each of the waypoints because the angle errors were larger than in other runs. For all three routes, each configuration of the model-based controller exhibits an inverse relationship between cross track error and the time taken to complete the route. The best setup for the most common scenarios that EOD robots will encounter is likely to be either Setup 3 or Setup 5, where the robot drives fast and stays fairly close the straight line path between waypoints. Those two setups have the desired goal heading, $\theta^{\star}$, set to be the angle from the previous waypoint to the current waypoint. This shows that the desired heading has more to do with the robot staying on the line between waypoints than do the gains.

Chapter 6

Conclusion

As shown in Chapter 5, the estimation and controls algorithms developed and implemented as part of this thesis resulted in significant improvements to the existing algorithms used for EOD robots. The Kalman filter showed a significant improvement over initial results. The state estimate improvement was due to a combination of fixing errors in the implementation of the Kalman filter and training the covariance values of the noise models.

The model-based controller was shown to drive at multiple velocities, across different surfaces, with no gain tuning required. This is contrary to the original PID controller. Additionally, the model-based controller exhibits smooth deceleration as it approaches waypoints, which is a consequence of the control law and not due to external commands to slow down like the PID controller. These improvements to robot navigation have been implemented in ACS and are now running on fielded systems to the benefit of end users.

There now exist better answers to the questions of "Where am I?" and "How do I get there?" due to the results obtained in this thesis.

6.1 Future Work

During the course of the research performed for this thesis several areas were not investigated as fully as they could have been. Other methods are also available that could offer further improvements to the state estimation and controls algorithms. Finally, areas other than those discussed in this thesis could benefit

from application of the results obtained here. These include:

- Use the Kalman filter learning algorithm, from Section 3.5.1, for indoor robots. The robots do not require a GPS sensor to benefit from the learning algorithm. If a suitable outdoor environment can be set up then DGPS can still be used as ground truth. A suitable environment could be a mock building without a roof. This would allow the DGPS receiver to get position measurements while simultaneously letting laser range finders have nearby objects to sense (assuming that the indoor robots are using lasers to perform some type of simultaneous localization and mapping algorithm (SLAM) that is common on current robots).

- Use a high quality IMU and compass, that serve as ground truth for Euler angles, in addition to the DGPS system so that the training algorithm attempts to minimize the errors between Euler angles in addition to position.

- Implement the more advanced work done by [Lapierre 06] and [Gulati 08] to improve the performance of the model-based controller. In particular, [Gulati 08] uses control Lyapunov functions that attempt to constrain the velocities and accelerations of the vehicle. They also use splines to generate intermediate waypoints between the vehicles current position and the goal position so that the route is smooth. Similarly, [Lau 09] uses quintic splines to generate routes.

- Extend the model-based controller to have constraints that allow the controller to work with non-differential drive vehicles such as those with Ackerman steering, like most personal automobiles [Shiller 91]. These other types of vehicles are not physically able to rotate in place.

- Build on the current extended Kalman filter in ACS to implement an unscented Kalman filter as described by [Thrun 06] and [Orderud 05] to better handle the nonlinearities of the system.

- Begin using the known outputs of the controller, or teleoperation commands, as inputs to the Kalman filter. This will change the system model in the

prediction update step (3.2) to

$$\hat{x}_{k+1}^{-} = \Phi_k \hat{x}_k^{+} + G_k u_k$$

where G_k is the control-input model that maps the control outputs to state variables.

Bibliography

[Abbeel 05] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y. Ng & Sebastian Thrun. *Discriminative Training of Kalman Filters*. In Proceedings of Robotics: Science and Systems, Cambridge, USA, June 2005.

[Ahuja 06] G. Ahuja, H.R. Everett, G. Kogut, E.B. Pacis & B. Sights. *An Adaptive Localization System for Outdoor/Indoor Navigation for Autonomous Robots*. In SPIE Proceedings 6230: Unmanned Systems Technology VIII, Defense & Security Symposium, April 2006.

[Aicardi 94] M. Aicardi, G. Casalino, A. Bicchi & A. Balestrino. *Closed Loop Smooth Steering of Unicycle-Like Vehicles*. In 33rd Conference on Decision and Control, pages 2455–2458, 1994.

[Aicardi 95] M. Aicardi, G. Casalino, A. Bicchi & A. Balestrino. *Closed Loop Steering of Unicycle-Like Vehicles via Lyapunov Techniques*. IEEE Robotics and Automation Magazine, vol. 2, pages 27–35, March 1995.

[Anderson 79] Brian D.O. Anderson & John B. Moore. Optimal Filtering. Dover Publications, First edition, 1979.

[Bruch 00] M. Bruch, R. Laird & H. R. Everett. *Challenges for Deploying Man-Portable Robots into Hostile Environments*. SPIE Proceedings: Mobile Robots XV, November 2000.

[Bruch 02] M. Bruch, G. Gilbreath, J. Muelhauser & J. Lum. *Accurate Waypoint Navigation Using Non-differential GPS*. AUVSI Unmanned Systems, July 2002.

[Congress 06] United States Congress. *Report to Congress: Development and Utilization of Robotics and Unmanned Ground Vehicles*. Technical report, Office of the Under Secretary of Defense, Acquisition, Technology and Logistics, Portfolio Systems Acquisition, Land Warfare and Munitions, Joint Ground Robotics Enterprise, October 2006.

[Ebken 05] J. Ebken, M. Bruch & J. Lum. *Applying Unmanned Ground Vehicle Technologies to Unamnned Surface Vehicles*. SPIE Proceedings: Unmanned Systems Technology VII, March 2005.

[Everett 02] H.R. Everett, R.T. Laird & M.R. Blackburn. *After Action Report to the Joint Program Office: Center for the Robotic Assisted Search and Rescue (CRASAR) Related Efforts at the World Trade Center.* Technical report, Space and Naval Warfare Systems Center, San Diego, August 2002.

[Gelb 74] Arthur Gelb, editor. Applied Optimal Estimation. The M.I.T. Press, First edition, 1974.

[Grewal 08] Mohinder S. Grewal & Angus P. Andrews. Kalman Filtering: Theory and Practice Using MATLAB. John Wiley and Sons, Inc., Third edition, 2008.

[Gulati 08] Shilpa Gulati & Benjamin Kuipers. *High Performance Control for Graceful Motion of an Intelligent Wheelchair.* IEEE International Conference on Robotics and Automation, pages 3932–3938, 2008.

[Hogg 02] R. W. Hogg & A. L. Rankin. *Algorithms and Sensors for Small Robot Path Following.* IEEE Conference on Robotics and Automation, May 2002.

[Kelly 94a] Alonzo Kelly. *A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles.* Technical Report CMU-RI-TR-94-19, Robotics Institute, Pittsburgh, PA, May 1994.

[Kelly 94b] Alonzo Kelly. *Essential Kinematics for Autonomous Vehicles.* Technical Report CMU-RI-TR-94-14, Robotics Institute, Pittsburgh, PA, May 1994.

[Khalil 02] Hassan K. Khalil. Nonlinear Systems. Prentice Hall, Inc., Third edition, 2002.

[Kim 05] Youngshik Kim & Mark A. Minor. *Bounded Smooth Time Invariant Motion Control of Unicycle Kinematic Models.* IEEE International Conference on Robotics and Automation, pages 3676–3681, April 2005.

[Lapierre 06] L. Lapierre, D. Soetanto & A. Pascoal. *Non-Singular Path-Following Cotnrol of a Unicycle in the Presence of Parametric Modeling Uncertainties.* In International Journal of Robust and Nonlinear Control, 2006.

[Lapierre 07] Lionel Lapierre, Rene Zapata & Pascal Lepinay. *Combined Path-following and Obstacle Avoidance Control of a Wheeled Robot.* International Journal of Robotics Research, vol. 26, no. 4, pages 361–375, 2007.

[Larson 06] J. Larson, J. Ebken & M. Bruch. *Autonomous Navigation and Obastacle Avoidance for Unmanned Surface Vehicles*. SPIE Proceedings: Unmanned Systems Technology VIII, Defense & Security Symposium, April 2006.

[Larson 07] J. Larson, M. Bruch, R. Halterman, J. Rogers & R. Webster. *Advances in Autonomous Obastacle Avoidance for Unmanned Surface Vehicles*. AUVSI Unmanned Systems North America, August 2007.

[Lau 09] Boris Lau, Christoph Sprunk & Wolfram Burgard. *Kinodynamic Motion Planning for Mobile Robots Using Splines*. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2427–2433, 2009.

[Micaelli 93] Alain Micaelli & Claude Samson. *Trajectory Tracking for Unicycle-Type and Two-Steering-Wheels Mobile Robots*, 1993.

[Nüchter 07] Andreas Nüchter, Giovanni Indiveri & Kai Lingemann. *High Speed Differential Drive Mobile Robot Path Following Control With Bounded Wheel Speed Commands*. IEEE International Conference on Robotics and Automation, pages 2202–2207, April 2007.

[Orderud 05] Fredrik Orderud. *Comparison of Kalman Filter Estimation Approaches for State Space Models with Nonlinear Measurements*. Scandinavian Conference on Simulation and Modeling, 2005.

[Phadke 99] Arun G. Phadke. Handbook of Electrical Engineering Calculations. CRC Press, First edition, 1999.

[Powell 08] Darren Powell, Mike Bruch & Gary Gilbreath. *Multi-Robot Operator Control Unit for Unmanned Systems*. Technical report, Space and Naval Warfare Systems Center, San Diego, August 2008.

[Rowe 08] Steve Rowe & Chistopher R. Wagner. *An Introduction to the Joint Architecture for Unmanned Systems (JAUS)*. Technical report, Cybernet Systems Corporation, May 2008.

[Rusu 05] Radu Bogdan Rusu & Marius Borodi. On Computing Robust Controllers for Mobile Robot Trajectory Calculus: Lyapunov. Unpublished technical report, 2005.

[Sakai 10] Atsushi Sakai & Yoji Kuroda. *Discriminatively Trained Unscented Kalman Filter for Mobile Robot Localization*. Journal of Advanced Research in Mechanical Engineering, vol. 1, no. 3, pages 153–161, 2010.

[Shiller 91] Zvi Shiller & Yu rwei Gwo. *Dynamic Motion Planning of Autonomous Vehicles*. IEEE Transactions on Robotics and Automation, vol. 7, pages 241–249, 1991.

[Simon 06] Dan Simon. Optimal State Estimation. John Wiley & Sons, Inc., First edition, 2006.

[Simon 10] Dan Simon. *Kalman Filtering With State Constraints: A Survey of Linear and Nonlinear Algorithms*. Control Theory and Applications, IET, vol. 4, no. 8, pages 1303–1318, 2010.

[Thrun 06] Sebastian Thrun, Wolfram Burgard & Dieter Fox. Probabilistic Robotics. The MIT Press, First edition, 2006.

[Ziegler 42] John G. Ziegler & Nathaniel B. Nichols. *Optimum Settings for Automatic Controllers*. ASME Transactions on Dynamic Systems, Measurement, and Control, vol. 62, pages 759–768, 1942.