AN OBSTACLE TRACKING AND CLASSIFICATION TESTBENCH FOR A MOBILE SENSOR PLATFORM

by

Brian W. Webb, B.Eng, Dalhousie University, 2002

A thesis submitted to the Faculty of Graduate Studies and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada K1S 5B6

January 2011

© Copyright 2011, Brian W. Webb
NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Avis:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.
Abstract

The GeoSurv II Unmanned Aerial Vehicle (UAV) project is focused on the construction of a UAV for geosurveying. It is intended to be able to fly with minimal operator interaction and as such must be able to detect, track and avoid obstacles automatically. Classically, obstacle detection and tracking are tightly linked to each other and to the sensors. This makes for an efficient system but poses certain problems if it becomes necessary to change one of the components, or to evaluate different approaches. The Obstacle Tracking and Classification (OTC) framework system developed in this research provides tracking and classification capabilities for a mobile sensor platform such as the GeoSurv II UAV and allows prediction and tracking methods to be evaluated and characterized so that an engineer can choose the most appropriate methods to solve their tracking problem. The OTC testbench system includes a scripted simulator for conducting the evaluation and characterization tests. A testing methodology and test scripts are presented and two prediction methods are tested as a demonstration.
Acknowledgements

I would like to thank Sean Bassett who laid the ground work for much of this research and Dr. Trevor Pearce who’s support and patience made this possible.
Dedication

To my family, without their help and support this would not have been possible. And most especially to my grandmother, Marjorie Kyte. Thank you for everything Baba.
Contents

1 Introduction ............................................. 1

2 Background ............................................. 3
   2.1 The GeoSurv II UAV Project .......................... 3
   2.2 Autonomy ............................................. 5
   2.3 Detection & Tracking .................................. 7
       2.3.1 Detection ......................................... 7
       2.3.2 Tracking ......................................... 7
   2.4 Uncertainty and Noise ............................... 8
   2.5 Control Systems ...................................... 9
       2.5.1 The Kalman Filter .............................. 10
       2.5.2 Other Kalman Filters ........................... 14
   2.6 Real Time Systems ................................... 16
   2.7 Inter-system Communication .......................... 17

3 State of the Art ......................................... 19
   3.1 Tracking Systems ...................................... 19
       3.1.1 EUCLIDE ......................................... 20
       3.1.2 Stiller et al ...................................... 20
       3.1.3 Sum of Absolute Differences ..................... 21
       3.1.4 Koyasu et al ..................................... 21
   3.2 Uncertainty & Prediction ............................ 22
       3.2.1 Filters ........................................... 22
   3.3 Object Classification .................................. 23
# Table of Contents

3.4 Confidence ........................................ 25
3.5 System Evaluation ............................... 26

4 The Thesis ........................................... 29
  4.1 Statement of Thesis ............................ 29
  4.2 Analysis ...................................... 30
  4.3 The Research .................................. 31
  4.4 Contributions ................................ 32

5 OTC System Design ................................. 34
  5.1 Operating Environment ......................... 35
  5.2 OTC Interface ................................ 36
    5.2.1 Measurement Units ....................... 37
    5.2.2 Received Data ............................ 38
    5.2.3 Published Data ........................... 41
  5.3 OTC Internal Structure ......................... 43
    5.3.1 Tracking and Matching .................... 46
    5.3.2 Prediction and Classification ............ 47
  5.4 Confidence ................................... 53
  5.5 Configuration ................................ 56

6 Simulation .......................................... 58
  6.1 Simulator Software ............................ 58
    6.1.1 The Noise Generator ..................... 60
  6.2 Simulated Noise ................................ 61
  6.3 Noise Validation ................................ 61

7 OTC Tests .......................................... 65
  7.1 Noise in the Tests ............................ 65
  7.2 Testing System ................................ 66
  7.3 Performance Measures and Tests ............. 67
    7.3.1 Accuracy ................................ 68
    7.3.2 Convergence ................................ 72

vii
List of Tables

7.1 Test standard noise ........................................... 67
7.2 KF: Prediction convergence. .................................. 73
7.3 EKF: Prediction convergence. ................................. 73
7.4 KF: Prediction times without noise. ......................... 75
7.5 KF: Prediction time with noise .............................. 76
7.6 EKF: Prediction times with no noise. ....................... 77
7.7 EKF: Prediction time with noise. ............................ 78
7.8 KF: Matching times without noise. ......................... 80
7.9 KF: Match times with noise. ................................. 81
7.10 EKF: Match times with no noise. ......................... 81
7.11 EKF: Match times with noise. ............................. 83
7.12 KF: Coverage variables. .................................... 84
7.13 KF: Coverage variables. .................................... 85
7.14 KF: OTC system output with a 40m gap. .................. 87
7.15 EKF: OTC system output with 40m gap .................... 88
7.16 KF: RMSE vs. noise levels ................................. 94
7.17 EKF: RMSE vs. noise levels ............................... 95
List of Figures

2.1 Autonomy software structure[1] ........................................ 5
2.2 Kalman Filter steps ...................................................... 11
3.1 Trajectory tree .......................................................... 22
5.1 Mobile sensor platform control environment ....................... 35
5.2 OTC system interface messages ...................................... 37
5.3 Coordinate conversion .................................................. 41
5.4 Block diagram of OTC system threads .............................. 43
6.1 Simulation toolchain .................................................... 59
6.2 Histogram of Gaussian noise with stdev 5m at 45.45 degrees north .. 62
6.3 Histogram of Gaussian noise with stdev 5m at 75.7 degrees west .. 63
6.4 Histogram of uniform noise on range measurements of 20m±2m .... 64
7.1 Obstacle and platform behaviour for accuracy tests .............. 69
7.2 KF: Prediction with no noise ....................................... 69
7.3 KF: Prediction with noise ........................................... 70
7.4 EKF: Predictions with no noise .................................... 71
7.5 EKF: Predictions with noise ........................................ 72
7.6 KF: Obstacles vs. prediction time without noise .................. 76
7.7 KF: Obstacles vs. prediction time with noise ...................... 77
7.8 EKF: Obstacles vs. prediction time without noise ................ 78
7.9 EKF: Obstacles vs. prediction time with noise .................... 79
7.10 KF: Obstacles vs. match time without noise ..................... 81
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.11</td>
<td>KF: Obstacles vs. average match time</td>
<td>82</td>
</tr>
<tr>
<td>7.12</td>
<td>EKF: Obstacles vs. match time without noise</td>
<td>82</td>
</tr>
<tr>
<td>7.13</td>
<td>EKF: Obstacles vs. match time with noise</td>
<td>83</td>
</tr>
<tr>
<td>7.14</td>
<td>Observation gap test</td>
<td>86</td>
</tr>
<tr>
<td>7.15</td>
<td>KF: 40m observation gap</td>
<td>87</td>
</tr>
<tr>
<td>7.16</td>
<td>EKF: 40m observation gap</td>
<td>88</td>
</tr>
<tr>
<td>7.17</td>
<td>Converging obstacles</td>
<td>89</td>
</tr>
<tr>
<td>7.18</td>
<td>KF: Converging obstacles</td>
<td>90</td>
</tr>
<tr>
<td>7.19</td>
<td>EKF: Converging obstacles</td>
<td>90</td>
</tr>
<tr>
<td>7.20</td>
<td>Sinusoidal obstacle motion</td>
<td>91</td>
</tr>
<tr>
<td>7.21</td>
<td>KF: Sinusoidal motion</td>
<td>92</td>
</tr>
<tr>
<td>7.22</td>
<td>EKF: Sinusoidal motion</td>
<td>93</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

An autonomous mobile sensor platform must operate in an environment with other objects such as vehicles, birds and buildings. Safe operation requires the ability to avoid these objects while still completing the assigned mission. Avoiding obstacles in turn relies on sensing and detecting the obstacles. The dynamic nature of moving objects requires a tracking method, since avoiding objects requires that their future positions be predicted.

There are many solutions to the tracking problem, but it can be difficult to compare these solutions because the object detection system and the tracking system are tied closely together. A general framework for tracking obstacles that is not highly dependent on the sensors and detector could be used to evaluate and compare tracking methods and provide the basis for the final tracking system used by a particular system.

The Obstacle Tracking and Classification system (OTC system) developed in this thesis demonstrates the development of such a framework, and how it can be used to gain a better understanding of the relative merits of the various tracking methods. This allows the creation of a general tracking tool which can act as a test bed for tracking methods, and can be used to quantify the attributes of such methods to determine their relative advantages and disadvantages.

A discussion of the relevant background material for autonomous, mobile sensor platforms is given in section 2 and an overview of the current state of the art with
CHAPTER 1. INTRODUCTION

regards to these systems is in section 3.

The research into such a tracking system is conducted in the context of the Geo­ 
Surv II UAV project at Carleton University. The tracker, the OTC system, requires 
only basic object information provided by the detection system and some typical 
vehicle position information. The current design is based on a publish/subscribe 
ar­chitecture, however; this is one of many methods that might be used to achieve in­ 
teroperability among components. The resulting loose coupling allows, the relatively 
independent development of the tracking system, the potential for the use of a wider 
range of sensor suites and detection methods, and the potential to reuse the tracking 
system across multiple platforms.

The development of autonomous capability must often be carried out before the 
target platform has been built, and on-board systems must be exercised in a realis­ 
tic way that does not rely on field-tests alone. To address this, a simulation-based 
testbench has been constructed as a flexible, reusable tool for exercising candidate 
tracking algorithms and quantifying their performance. The design of the testbench 
is discussed in section 5 and the simulation tools are covered in section 6.

The testbench is organized around a standard set of tests, and also allows cus­ 
tomized tests to be introduced easily. Each test is a scenario in which the motions 
of the platform and objects are specified. The testbench allows controlled noise to be 
introduced into both the positional awareness and object detection data fed to the 
tracking system. The testbench allows the performance of the tracking prediction 
methods to be measured in five ways: the accuracy of the predictions, the precision 
of the predictions, the amount of time needed to make the predictions, the amount 
of computing resources needed to make the predictions and the sensitivity of the pre­ 
diction method to noise in the measurements. The testbench is demonstrated with a 
Kalman filter (KF) and an Extended Kalman filter (EKF). The tests and the results 
are presented in section 7.
Chapter 2

Background

There are a number of concepts that need to be considered when designing and building an obstacle tracking system for an autonomous sensor platform such as the GeoSurv II UAV. A platform intended to operate with minimal input from the operators requires robust autonomy software. It must be capable of making the correct decisions to ensure both platform and bystander safety, and to complete the designated mission. To do this the autonomy software must be able to localize the platform itself and detect and track obstacles in its operating environment. Sensor noise and measurement uncertainty must also be dealt with to ensure information is accurate enough to make appropriate decisions. Control systems theory provides methods for handling noise and uncertainty as well as for tracking obstacles. To avoid those obstacles and manoeuvre the platform safely, the autonomy software must make its decisions in real time. Finally, for all of the systems on the platform to work correctly they must be able to communicate with each other.

2.1 The GeoSurv II UAV Project

In 2004 Sander Geophysics Limited approached Carleton University about researching the design and construction of an UAV for geomagnetic surveying. Geomagnetic surveying involves measuring the changes in the Earth's magnetic field to determine the geological features of the terrain being overflown. It can be used to de-
CHAPTER 2. BACKGROUND 2.1. THE GEOSURV II UAV PROJECT

tect various kinds of rock, including ore bearing deposits. The surveys are performed using sensitive instruments known as magnetometers, and are usually conducted by aircraft.

Fixed wing aircraft allow for relatively rapid surveying, but must fly at heights that reduce the resolution of the survey for safety reasons. The closer to the ground the survey is performed, the higher the resolution possible. Fixed wing aircraft also cannot follow changes in terrain altitude closely. This can obscure interesting geological features and reduce the utility of the survey. Helicopters can fly surveys at much lower altitudes and follow the terrain more accurately than fixed wing aircraft, but are much slower and more expensive to operate. Both types of aircraft also introduce error into the survey measurements. Their airframes and equipment, such as motors, avionics and radios contain materials with magnetic signatures that produce varying electromagnetic fields that can interfere with the magnetometers.

Sander Geophysics Limited is interested in the possibility of using a fixed wing UAV to conduct surveys. The UAV must be able to fly at much lower altitudes than a manned aircraft and follow the terrain more accurately than a human pilot. It must also use non-magnetic materials in its construction. Ideally it could reduce the cost of operations by requiring fewer ground crew for operation and maintenance, and open up new territory for surveying by not requiring a nearby airport.

The Mechanical and Aerospace Engineering and Systems and Computer Engineering departments at Carleton agreed to undertake this research. Developing a composite materials airframe with a low magnetic signature, evaluating survey results and costs and developing control systems for the UAV, dubbed GeoSurv II. The control systems, specifically the autonomy software, are intended to provide GeoSurv II with a high degree of operator independence. Both to reduce operator load and fatigue and to allow GeoSurv II to survey under radio silence.

The basic structure of GeoSurv II's autonomy system was developed by Sean Basset[1]. Shown in figure 2.1, the autonomy system has a hierarchical structure with three layers. The highest layer is the deliberator, it contains the route planner which determines the UAV's flight path based on information such as the mission parameters and obstacle data. A flight path is a sequence of way-points, The middle layer, the sequencer, generates response commands based on the flight path way-
CHAPTER 2. BACKGROUND

2.2. AUTONOMY

points and on information from the lower layer. The lowest layer is the controller layer. It contains the autopilot. The autopilot executes low level operations that control the UAV’s flight.

![Autonomy software structure](image)

Figure 2.1: Autonomy software structure[1]

All three layers of the autonomy system require information about the UAV platform and about the environment to operate correctly. The research covered in this document concerns itself with determining how to provide accurate estimations of obstacle positions and motion to the other parts of the autonomy system. This is a part of the deliberator layer.

2.2 Autonomy

An autonomous sensor platform can operate in conditions where a remotely controlled platform cannot. An autonomous vehicle can operate out of line of sight with its operator, beyond practical communications range, and does not need to be in constant
contact with the operator. Robust autonomy also allows the platform to operate in conditions and situations where communication with the operator is not possible.

To determine its location, a sensor platform will commonly use localization systems such as the Global Positioning System (GPS) and Inertial Navigation (or Measurement) Units (IMUs). However, relying on only self-localization and prior environment maps is not sufficient to ensure safe operation of an autonomous sensor platform [2]. Environment sensing is also necessary for autonomous operation in order to avoid obstacles and hazards, track targets of interest and locate landmarks. This is especially true when the platform is out of contact with its operators[3].

Additionally, information from localization systems may not allow the sensor platform to localize itself with sufficient accuracy for its purposes[3]. For example, commercial GPS has an uncertainty of ±6.7 meters and is subject to noise[4]. Inertial systems suffer from accumulated error since they work by adding detected changes to the system state and any error, even very small error, will accumulate[5]. This leads to 'drift' in the predicted location of the platform.

The techniques used in robotics and autonomous vehicles for determining location and constructing a map of the operating environment are known collectively as Simultaneous Localization and Mapping (SLAM). SLAM techniques use information from the platform's sensors to detect and identify landmarks which are then used to identify the platform's location. Landmarks and other information are also stored by the platform as maps.

The maps used by SLAM techniques may be \textit{a priori} maps, which the platform uses to verify its location by matching landmarks, and which it updates using gathered data. Alternatively the maps may be constructed purely as the platform travels and explores its environment.

Therefore, for safe, reliable autonomy a mobile sensor platform must be able to gather information about its environment to assist in localization and to allow it to plan actions and avoid dangers.
2.3 Detection & Tracking

Automated object detection and tracking is commonly carried out in several different fields. One of these fields is the control of unmanned vehicles such as those involved with the DARPA grand challenge in 2006 and 2007[3]. Another field is providing enhanced information and automated guidance for human controlled vehicles, such as with ACC[6].

2.3.1 Detection

Obstacle detection is an important topic in regards to the control of autonomous vehicles. However, detection can be assumed to work when dealing with tracking and classification. The detector separates the obstacle information from the background information in the sensor data and presents this data to the rest of the control software as a set of observations, one observation per obstacle, one set per update period. Accordingly, the research presented in this thesis does not deal with detection in detail.

It should be noted that obstacle detection methods and sensors are very closely related. For example, stereoscopic image analysis requires a pair of cameras arranged in a specific geometry. This means that the sensors and obstacle detector can be treated as a single system in most cases.

2.3.2 Tracking

Once an object has been detected it is usually useful to keep track of it, so that predictions about its behaviour and intelligent responses to its presence can be made if necessary. Tracking requires matching an object from one set of observations to the next. This depends on the motion of the object and the motion and update rate of the sensors, and requires predicting where the object will likely be when the next set of observations is made. Filtering and other prediction methods are used to determine the likely future state of the object for matching algorithms to work with[7]. In state space models this is done by estimating the object's state from measurements and then using the physical model of the object to project those state's forward in time...
to the next sampling interval[6]. A common method for handling such filtering and prediction is the Kalman filter and its descendents, covered in section 2.5.1.

Tracking is a complex task because of a number of different factors[8]:

- Loss of information in the sensor technology, for example projecting from a 3D world to a 2D image by camera based sensors,
- Noise in sensor data,
- Complex motion of the tracked object,
- Nonrigid or articulated nature of tracked objects,
- Partial or full object occlusions,
- Complex object shapes,
- Scene illumination changes or other environmental factors, and
- Real-Time processing requirements.

There are many different methods for matching new observations to known obstacles. Some rely on pure position comparisons, i.e. how close the observed obstacle is to where the known obstacle is[9]. Other methods add in obstacle shape, size or other information such as temperature or colour. This often results in object detection and tracking being combined into a single system with both the detection and tracking methods tied to the kinds of sensors used [10, 9, 11].

### 2.4 Uncertainty and Noise

Knowledge of an actual system is never perfect. There is always an amount of uncertainty in the model of a system’s state. In a digital control system there are three sources of uncertainty. The two most obvious being the quality and capabilities of the transducers used to measure the state of the system, and the granularity of the analog to digital (A/D) converters used to convert transducer signals into digital information. For example uncertainty in measurements from a hyperboloidal mirror
based panoramic camera comes from quantization errors, and from a blurring effect caused by the panoramic conversion[2].

The final source of uncertainty is model uncertainty [12]. Generally, a system is too complex to model completely. It may not be entirely understood, it may take too long to develop a model with sufficient fidelity, or such a model would require more computational resources than are available. Therefore the model that is actually implemented and used is usually a simplified representation of the actual system. This reduction in complexity makes it easier to implement the model, and causes it to consume fewer resources, but comes at the cost of increased uncertainty. The simplified model may provide inexact, or even incorrect, answers in situations which fall outside of the assumptions made when the simplification was performed.

The uncertainty in measurements from a transducer caused by random phenomena in the transducer system is called noise. The phenomena responsible for noise can be characterized statistically if they are static, not changing with time[13]. As such, different kinds of noise can be classified according to their characteristics, allowing the random process that produces the noise to be described.

There are many statistical methods for describing random noise. One of the most common is the probability density function (PDF)[13]. The PDF of a random variable is a function that describes the likelihood that the random variable will have a specific value. The PDF of the most common kind of naturally occurring random phenomena is the Gaussian, or normal PDF[13, 14, 15]. The most likely values for this PDF are in the center, near the median value, and when plotted this gives the Gaussian PDF its characteristic bell shape.

## 2.5 Control Systems

Control systems engineering is concerned with the understanding and control of parts of the engineer’s environment, called systems. Systems are interconnections of elements and devices which are created to achieve a purpose[16]. It is possible for naturally occurring systems to exist, but as the engineer has no control of these without adding devices, they are not generally considered in control systems theory.
A dynamic system is a system which changes with time, and can usually be described by ordinary differential equations. Most physical systems are non-linear and dynamic, and are modelled linearization[16] to simplify solutions. However, this simplification introduces model uncertainty and can produce inexact results.

The purpose of control systems theory is to provide and develop reliable, quantifiable methods for controlling processes and systems so that a desired outcome is reached. Much of control systems theory is devoted to working with systems where the available information is either incomplete, uncertain, noisy or a combination of all three[16].

One common method for handling these three problems is the use of filters. Filters are mathematical algorithms used to extract information from noisy or incomplete data[17]. They are called filters because, analogously with physical filters and electronic signal filters, they extract useful information from a noisy source[15]. Filters belong to a class of techniques known as least-squares[5], as they are descended from the least-squares estimation method developed by Carl Friedrich Gauss in 1795[17].

Filters are not the only method of system state estimation in control theory, other least-squares techniques such as regularization and uncertainty ellipsoids, as well as probabilistic techniques such as Bayesian reasoning and evidence statistics can also be used. However, filters are widely used because of their ease of implementation, and the general accuracy of their results[17, 18].

Another common use of filters is sensor fusion which combines measurements taken from several different sensors into a single observation. This approach is used on projects from robotics control[5] to adaptive cruise control(ACC)[19].

### 2.5.1 The Kalman Filter

One of the most basic, and most common filter in control systems theory is the Kalman filter, which is also known as linear quadratic estimation. The KF is an estimator for the linear-quadratic-Gaussian problem, which is the problem of estimating the instantaneous state of a linear dynamic system perturbed by Gaussian white noise[17]. More simply, it is a recursive algorithm that estimates the state of a linear dynamic system from a series of noisy (white noise) measurements[15]. This in essence means...
the Kalman filter is an optimal one-step-ahead predictor for observed data[20].

The KF is so common in practice because it is a statistically ideal estimator[17] and because it is easy to convert into a computer program. This conversion is easy because the KF uses a finite representation of the estimation problem, with a finite number of variables[17, 18].

The algorithm was developed between 1958 and 1961 independently by Rudolf E. Kalman, Ruslan L. Stantonovich, Thorval Nicolai Thiele and Peter Swerling. The first known implementation of the filter was by Stanley F. Schmidt, and was used in the trajectory estimation and the navigation computer of the Apollo program[17].

The basic Kalman filter is a recursive estimator which means that only the last state estimate (the prediction) and the current state observations are needed. No other information must be tracked and no history of the predictions beyond the last needs to be kept. The filter’s state is tracked by a vector, $X_{k|k}$: the a posteriori state at time $k$ given observations up to and including time $k$, and by a matrix, $P_{k|k}$: the a posteriori error covariance matrix (measure of the estimated accuracy of the state estimate)$^1$.

\[ X_{k-1|k-1} \xrightarrow{\text{Predict Phase}} X_{k|k-1} \xrightarrow{\text{Update Phase}} X_{k|k} \]

Figure 2.2: Kalman Filter steps

Figure 2.2 shows that the basic filter algorithm has two phases, predict and update. The predict phase uses the state estimate from the previous time step ($x_{k-1|k-1}$) to produce an estimate for the current time step ($x_{k|k-1}$). This is the a priori state estimate, so called because while it is an estimate of the state at the current time

$^1$The subscript notation has three commonly seen variations:

- $k - 1|k - 1$ At time $k - 1$ given observations up to and including time $k - 1$.
- $k|k - 1$ At time $k$ given observations up to and including time $k - 1$.
- $k|k$ At time $k$ given observations up to and including time $k$. 

11
CHAPTER 2. BACKGROUND  2.5. CONTROL SYSTEMS

it does not yet include the current measurements. In the update phase, the a priori estimate is combined with the current measurements to produce the a posteriori estimate \((x_{k|k})\) and the error covariance. The filter’s gain is also updated in this phase, towards a more optimal value.

The symbols used in Kalman filter equations are\([17, 18]\):

- \(x\) State vector of a linear dynamic system. Contents and size depend on the system under consideration.

- \(z\) Vector, or scalar, of measured values. These are what the sensors report about the state of the system.

- \(u\) The control input vector. Supplied by a system’s user to adjust the output of the system. If no control input is applied to the system by the user this term is zero and can be ignored.

- \(w\) Process noise. Noise inherent to system, including control input noise.

- \(v\) Measurement noise. Noise introduced by the sensors or measurement methods.

- \(y\) Innovation (or measurement) residual. Reflects the difference between the prediction and the actual observation. Is zero when they are equal.

- \(S\) Innovation (or residual) covariance.

- \(R\) Covariance matrix of observational (measurement) uncertainty.

- \(Q\) Covariance matrix of process noise in the system state dynamics.

- \(\Phi\) Covariance matrix of state estimation uncertainty.

- \(K\) Kalman gain matrix. The filter’s gain values for each state.

- \(H\) Observation or measurement sensitivity matrix. Defines the linear relationship between the state of the dynamic system and the measurements that can be made.

- \(B\) Input state space mapping.

12
• $F$ Coefficient matrix of a continuous linear differential equation defining a dynamic system.

The state of a system and the measurements in discrete time are represented by:

$$x_k = F_k x_{k-1} + B_k u_k + w_k$$  \hspace{1cm} (2.1)

$$z_k = H_k x_k + v_k$$  \hspace{1cm} (2.2)

Where $w_k \sim N(0, Q_k)$ is the Gaussian noise with covariance $Q_k$ and $v_k \sim N(0, R_k)$ is Gaussian noise with covariance $R_k$.

The two phases, and the steps within each phase, of the KF must be performed in the correct order. The first phase is the predict phase which generates, in order, the \textit{a priori} state prediction ($x_{k|k-1}$) and then the \textit{a priori} estimate covariance ($\Phi_{k|k-1}$). The second phase is the update phase which generates the innovation residual ($y$), the innovation covariance ($S_k$), the Kalman gain ($K_k$), the \textit{a posteriori} state estimate ($x_{k|k}$), and the \textit{a posteriori} estimate covariance ($\Phi_{k|k}$).

The equations of the discrete time Kalman filter phases are [17]:

**Predict**

Predicted (\textit{a priori}) state

$$x_{k|k-1} = F_k x_{k-1|k-1} + B_k u_k$$  \hspace{1cm} (2.3)

Predicted (\textit{a priori}) estimate covariance

$$\Phi_{k|k-1} = F_k \Phi_{k-1|k-1} F_k^T + Q_k$$  \hspace{1cm} (2.4)

**Update**

Innovation residual

$$y_k = z_k - H_k x_{k|k-1}$$  \hspace{1cm} (2.5)

Innovation covariance

$$S_k = H_k \Phi_{k|k-1} H_k^T + R_k$$  \hspace{1cm} (2.6)
Kalman Gain

\[ K_k = \Phi_{k|k-1} H_k^T S_k^{-1} \]  \hspace{1cm} (2.7)

Updated (a posteriori) state estimate

\[ x_{k|k} = x_{k|k-1} + K_k y_k \]  \hspace{1cm} (2.8)

Updated (a posteriori) estimate covariance

\[ \Phi_{k|k} = (I - K_k H_k) \Phi_{k|k-1} \] \hspace{1cm} (2.9)

The innovation residual \( y_k \) reflects the discrepancy between the predicted measurement \( H_k x_{k|k-1} \) and the actual measurement \( z_k \). If the residual is zero, then the two are in complete agreement.

### 2.5.2 Other Kalman Filters

There are a number of variations of the Kalman filter, intended to address the shortcomings of the basic version, mostly the assumptions of linearity and Gaussian noise. The most common variation on the KF is the Extended Kalman filter (EKF).

The EKF was created to attempt to deal with non-linear systems. It is implemented by linearizing about the current mean[17].

The state transition and observation functions do not need to be linear functions of the state but may be differentiable functions instead. The state and measurement equations of the EKF are similar to those for the Kalman filter (2.1,2.2), as seen here:

\[ x_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \] \hspace{1cm} (2.10)

\[ z_k = h(x_k) + v_k \] \hspace{1cm} (2.11)

Where \( w_k \) and \( v_k \) are the process and observation noises which are assumed to be zero mean Gaussian noises with covariances \( Q_k \) and \( R_k \) respectively. The functions \( f \)
and \( h \) can be used to compute the predicted state and predicted measurement from the previous estimate. However, they cannot be applied directly to the covariance and the Jacobian (a matrix of partial derivatives) must be computed. It is this process which linearizes the non-linear system about the current estimate[17].

Once the linearization has been performed the rest of the filter process is handled using the same algorithm as the basic Kalman filter. However the state transition and observation matrices are now defined by the following Jacobians:

\[
F_{k-1} = \frac{\delta f}{\delta x} \bigg|_{\dot{x}_{k-1}|k-1,u_{k}} 
\]

(2.12)

\[
H_k = \frac{\delta h}{\delta x} \bigg|_{\dot{x}_{k}|k-1} 
\]

(2.13)

The predict and update phases of the EKF are the same as those of the KF, with changes to the equations for the \emph{a priori} state in the predict phase and the innovation residual in the update phase because of the linearization.

**EKF \emph{a priori} state**

\[
x_{k|k-1} = f(x_{k-1|k-1}, u_{k-1})
\]

(2.14)

**EKF innovation residual**

\[
y_{k|k-1} = z_k - h(x_{k|k-1})
\]

(2.15)

The EKF has some disadvantages. It is not generally an optimal estimator[21], unless the measurement and state transition model are linear. In which case it behaves exactly as the basic filter. The EKF may also diverge from the actual system if the initial estimate is too far from the actual value, or if the process is modelled incorrectly because of the linearization. The estimated covariance matrix also tends to underestimate the actual covariance matrix and can become statistically inconsistent unless stabilizing noise is added[21]. Despite its drawbacks the EKF has been a \emph{de facto} standard for navigation and global positioning systems (GPS) for many years[21].
2.6 Real Time Systems

A real time system (RTS) is a computer system that is capable of dealing with events and signals received from the outside world in a manner so that timing deadlines are met. There are two kinds of deadlines that real time systems deal with, hard deadlines which must be met absolutely, and soft deadlines which ought to be met, but in practice can be missed. A system that deals only with hard deadlines is called a hard real time system, while a system that does not is a soft real time system.

A useful autonomous mobile sensor platform must be able to process the input from its sensors and make the decision to evade an obstacle before the obstacle is reached. The unforgiving nature of failing to make such an evasion means that mobile sensor platforms should be classed as hard real time systems. If the platform’s mobility parameters (such as maximum speed, turning radius, acceleration and deceleration) and its sensor parameters (such as range and update rate) are known, the longest amount of time obstacle detection and decision making can take can be determined. For example, if the platform is moving at a constant 10 meters per second and there is an obstacle 20 meters ahead, it must, detect and respond to that obstacle within 2 seconds. If the sensor’s update rate is once every 0.5 seconds and the platform’s quickest evasive manoeuvre takes 0.5 seconds, then detection and decision must take less than 1 second to perform. In the worst case there is a 0.5 second delay between when the obstacle can be observed and when the sensors report it. In addition, if sensor data is not processed faster than it arrives, for example if the update rate is once every 0.5 seconds and the processing takes 0.75 seconds, information may be lost and incorrect decisions may be made.

Three things determine the speed at which a process responds to events. The total number of processor cycles required to compute results and produce the response, the frequency of the processor, and the likelihood of the process being interrupted. Thus to determine if an autonomous sensor platform can safely respond to events in the world around it, the performance of all processes and systems running on its computers must be measured. This is a basic analysis of feasibility for any proposed autonomous sensor platform design.
2.7 Inter-system Communication

An obstacle tracking and classification system must interact with other systems on the mobile sensor platform. It must be able to receive information from, and report results to, the other systems being used by the mobile sensor platform. There are a number of potential communication methods such as shared memory, message passing and publish/subscribe architectures.

A publish/subscribe architecture middleware has several advantages for use in a mobile sensor platform. In a publish/subscribe architecture, messages are generated by publishers and consumed by subscribers. The publisher needs to know no details about the subscriber or subscribers. It simply indicates that it will be publishing a message of a certain type or with a certain descriptor and then begins producing the messages. Subscribers also need to know nothing of message publishers. Subscribers only indicate that they are interested in receiving messages of certain types or with certain descriptors. Once this is done the subscriber begins to receive messages, but only those messages in which it is interested. The decoupling of the message generators and consumers allows for increased scalability of the system. Any number of subscriber processes can access the published data and the number of subscribers can change, even during run time. As a result, the system is modular and sections of it can be modified without having to adjust all of the other parts.

A publish/subscribe middleware can also be used to distribute the system across multiple processors and physical computer systems. If one part of the system using the middleware is so resource intensive that it is not practical to have it share the same physical system as other parts of the software system, communication between the two or more computers should be as transparent as possible. A well designed middleware should make the inter-operation of different systems as simple as inter-operating on a single computer system.

The Inter-Process Communication (IPC) middleware[22] from Carnegie Mellon University provides publish/subscribe functionality as a platform independent library. It is a network based architecture and supports both the publish/subscribe and client/server models of communication. It also supports the passing of data structures between programs written in different programming languages, although only
C/C++ and LISP are supported in version 3.6.
Chapter 3

State of the Art

In this section an overview of four tracking systems is presented, showing how detection and tracking are performed. This is followed by a discussion of the handling of noise and uncertainty using filters and then by an examination of classification methods and of classification and tracking confidence. Finally, metrics for evaluating tracking system performance are introduced.

3.1 Tracking Systems

Tracking systems are a well developed technology, however; there are no true standards for the evaluation of different tracking systems and methods[9]. This is the result of a lack of common data sets and even performance measures. Different systems using different sensors cannot necessarily use the same data sets or retrieve the same amount of useful information from the same data set. This is most often the case when very sensor specific information, such as colour, is used when matching observations to known obstacles. Also different tracking systems will have different goals and requirements. For some systems, differentiating among obstacles is very important, while other systems may merely be concerned with knowing that the obstacles are present.

Four tracking systems are examined briefly to illustrate how tracking can be handled.
3.1.1 EUCLIDE

The EUCLIDE tracking system[19] is a component of an Automatic Cruise Control (ACC) for road vehicle applications. The system uses a combination of Kalman filters to perform data fusion on inputs from a variety of sensors, including infrared cameras and lidar, and estimates the velocity of each obstacle detected. From this estimate, the obstacle’s future position can also be estimated and compared against future measurements to decide whether an obstacle was detected previously.

The EUCLIDE system found that simple filters, and single kalman estimators performed poorly. Therefore, adaptive estimators were designed by combining multiple filters to achieve acceptable estimation and improved tracking performance.

EUCLIDE also deals with tracking road borders. It treats each road segment as a curve whose curvature is linearly dependent on its arc length. This allows EUCLIDE to treat road borders as 3rd order polynomials. Measurements of the road borders are passed through filters which use these polynomials as their model.

Each obstacle detected by EUCLIDE produces an estimate of its position. Each position is recorded, and a series of positions that are grouped together, progressing through time in parallel with the tracked road borders are considered an obstacle’s trajectory. The trajectories are also passed through filters to produce better estimates of the obstacle’s velocity and motion.

3.1.2 Stiller et al

Another ACC tracking approach, proposed by Stiller et al[6], uses both a position estimate and geometric features, such as corner points to compare new observation data to predicted obstacle locations. The addition of the geometric features helps stabilize the tracking and increase match confidence. In this approach, objects are divided into segments, and the segments are associated with obstacles based on a distance measure between observed measurements and predicted state. The threshold distance used to separate and identify segments from different obstacles is determined a priori, not dynamically, and can require tuning.
CHAPTER 3. STATE OF THE ART

3.1. TRACKING SYSTEMS

3.1.3 Sum of Absolute Differences

The Sum of Absolute Differences (SAD) is another matching algorithm, useful for image-based comparisons and can be used when comparing small sections of images or entire images. To perform SAD an region of interest in one image is defined, for example the portion that contains an obstacle. Then another image is examined by dividing it up into blocks based on the size and shape of the original region of interest. The region of interest is subtracted from each of the blocks and second, the block is considered to match the region of interest if the sum of the absolute differences meets two criteria. First the block's SAD must be the smallest out of all of the blocks, and the SAD must be lower than a given threshold. The threshold is highly application, and even environment specific[2]. As a camera based technique, SAD matching depends on the camera resolution, camera colour depth, and the number of cameras as well as lighting and other environmental factors. It also requires large amounts of computing resources.

3.1.4 Koyasu et al

To deal with uncertainty in tracking, it is possible to use predictions to generate potential obstacle trajectories, and trajectory trees. A trajectory tree is a collection of potential trajectories, each of which is associated with the likely future trajectories the obstacle could have if the original trajectory is correct[2]. Figure 3.1 illustrates the conceptual model of a trajectory tree.

The true obstacle trajectory, and thus an accurately predicted position, is built up over time by eliminating trajectories from the trees that do not persist. A tracking system using this method uses estimated velocity to distinguish between static and moving obstacles. Obstacles with consistently low estimated velocities are considered to be static, and any apparent motion is assumed to be the result of sensor noise and jitter.
3.2 Uncertainty & Prediction

There are a number of mathematical tools available to help deal with uncertainty and noise in sources of data. Probability theory is one of the earlier methods[17] but other methods have been developed, such as fuzzy sets, possibility theory, the theory of fuzzy measures and evidence theory[7].

Additionally some methods for dealing with noise and uncertainty can also be used as state predictors. The Kalman filter and its derivatives are useful for being able to handle both problems.

3.2.1 Filters

To perform prediction with a filter, the target object who’s future state is being predicted is modelled using the physical laws of object motion[7]. The object detector and sensor systems report information on the object’s state, by either observing the state directly, or by observing variables from which the state can be derived.

A common representation of the states of a simply moving object includes the position \((x, y, z)\), velocity \((\dot{x}, \dot{y}, \dot{z})\) and acceleration \((\ddot{x}, \ddot{y}, \ddot{z})\) of the object[7, 6]. In cases where constant velocity can be assumed, the state space can be simplified to include just the position and velocity[2].

The filter’s reaction time and performance can vary depending on the state vari-
ables modelled. For example, a filter-based sensor fuser and object tracker[7] achieved better reaction time by modelling angular acceleration as opposed to angular velocity, but at the cost of stability.

One common modern application of filter and prediction involves ACC systems, such as the EUCLIDE system described in section 3.1.1. These systems use data gathered from one or more cameras to adjust a car’s speed relative to traffic and sound collision warnings. Data fusion, object tracking and noise filtering are all performed by filters [19, 11], though the exact filter depends on the design requirements of the ACC system.

### 3.3 Object Classification

Once an object has been detected it is often useful to classify it in a category to help with decision making. Objects can be classified in a number of different ways which tend to be highly dependent on the characteristics of the objects associated with the mission being performed by the detecting platform. For instance, an ACC system might have the classifications Vehicle, Sign, Road Marker, Pedestrian and Other, while a UAV might have the classifications Building, Aircraft, Bird and Cloud.

Classification is important for decision making. An off-road wheeled vehicle should be able to distinguish between obstacles which can potentially damage it and ones which will not. A large rock presents a different problem than does a large tuft of grass[3].

Classification depends on what information the available sensors can provide, and what features can be distinguished. Black and white cameras cannot provide information on object colour beyond how light or dark it is. Radar or sonar can provide even less information on colour, and likely little information on the shape of the object. The choice of features to use for classification is critically important to the performance of a classifier[23].

One method of classification[7] is performed by creating a series of templates, which contain information describing a class of object. Objects detected by the sensors are compared to the templates and if there is a ‘close enough’ match with one
of them, then the object is considered to be of the class that the matching template describes. The definition of 'close enough' is dependent on the needs of the particular system, and the capabilities of the sensors on that system.

Another method, discussed earlier in 2.3.2, uses estimated velocity to distinguish between static and mobile obstacles. If the velocity is consistently below a set threshold for a set amount of time, the obstacle is considered to be stationary[2].

A minimum error classifier is a classification algorithm that produces the lowest possible probability of misclassification. A minimum error classifier is known as a Bayes classifier, and the minimum probability of misclassification is known as the Bayes error[23]. Bayes classifiers are not necessarily implemented, for reasons of computation cost and high information requirements[23]. Other algorithms which approximate Bayes classifiers are often used instead. Including the k-Nearest Neighbour (k-NN) Classifier, the Maximum Likelihood Classifier and the Support Vector Machine Classifier[23].

To explain how the k-NN Classifier works two terms must first be defined, feature and feature space. A feature is a measurable property of the obstacle, such as colour, shape or size. A feature space is an abstract, n-dimensional space that represents all possible combinations of features. The size of n is dependent on the number of features being considered. The k-NN Classifier is trained by having it examine the features of a set of training obstacles where the classification of all the obstacles is known beforehand. Once trained a k-NN classifier categorizes obstacles by assigning them to the class of their k of nearest neighbours in the feature space. The k nearest neighbours are from the training data set.

If k=1 the k-NN Classifier is simply known as a Nearest Neighbour classifier. However, if k=1 the minimum probability of misclassification approaches twice the Bayes error. Increasing k reduces the probability of misclassification but as k increases so do the memory and processing requirements. This means that the suitability for implementation of the k-NN Classifier decreases as accuracy increases.

Both the maximum likelihood classifier and the support vector machine view data points as p-dimensional vectors. For a given implementation of these types of classifiers the vector space they use will cover all the possible kinds of information available. Both the maximum likelihood classifier and the support vector machine require
training to produce useful classifications. To train a maximum likelihood classifier or support vector machine at least one set of training data must be created. This set, or sets, of training data are referred to as training vectors.

In the maximum likelihood classifier the distribution of training vectors from the same class is modelled as a mix of Gaussian density functions. Model parameters are identified by applying an appropriate clustering algorithm, or the Expectation-Maximization algorithm, to the training vectors of each class.

A support vector machine is a linear classifier operating in higher dimensional space. A set of non-linear transformations map an N-dimensional input vector into an M-dimensional feature space where $M > N$. A linear classifier, consisting of $M$ weights operates on the M-dimensional space, with the optimal weights for the classifier represented by a subset of the training vectors.

The performance of both the maximum likelihood classifier and the support vector machine algorithms is highly dependent on the quality of the training vectors available. Poor training vectors, or poor selection of training vectors will decrease the accuracy of both of these algorithms [23].

### 3.4 Confidence

There is the possibility of errors being made when classifying a detected object or matching a newly observed object position to a predicted one. For this reason it is often desirable to attach a confidence value to a prediction or a classification [7]. Confidence values are usually a representation of how likely an output result is to be correct.

Sensor performance is dependent on external factors that the designer of the system cannot control, for example weather conditions, or traffic in the case of systems designed to drive on public roads [6]. Confidence values can be assigned directly to measurements, to indicate the sensors own limited capabilities when unfavourable conditions are detected [6]. This is referred to as self-assessment, and is a key component of designing reliable systems.

One reliability measure used by the stereo vision sensors of an automated driving
3.5 System Evaluation

Comparison between various tracking algorithms and methods is often done using simulated data, and although there is usually a trade-off between computational and development resources and the complexity of the simulation model, the results of simulation studies are useful in comparing various methods. However, exhaustive testing of all of the parameters that can affect the performance of a tracking method is difficult, and this can lead to incorrect conclusions being drawn[10].

Some of the more common issues that should be considered when designing and evaluating target tracking algorithms are: false track determination, long term track retention, algorithm reliability/robustness, and track estimate/prediction accuracy. These are not the only possible issues, and the priority of the issues depends on the application[10].

A sensor return may be an actual object, and if the trajectory track developed for the object is a reasonably accurate representation of the object’s motion, the track is considered to be true. If the object is the result of sensor noise or clutter, or if the trajectory track is not an accurate representation of the object’s motion, the track is considered to be false. Determining if a track is true or false, or if it has become true or false is known as false track determination. Long term track retention is a measure of a tracking algorithm’s ability to correctly identify and track an object over a series of observations.

One issue with comparing various methods of object tracking is that there tends to be a lack of uniformity in evaluation methods[9]. This is the result of a lack of common data sets and measures of performance between independently developed and implemented algorithms. Some measures of a good tracker are: the ability to
track and identify objects well, that is tracking individual objects consistently over a
long period of time; tracking objects in spite of distractions (occlusion, illumination
changes, etc.); accurately estimating object parameters (such as object velocity, ac-
celeration); and the ability to perform tracking and prediction in the available time[9].

The differences between predicted values produced by an estimator and the actual
values is known as the forecast error[14]. There are several ways of measuring the
forecast error of an estimator. The Root Mean Squared Error (RMSE) shown in
equation 3.1 is one such measure. The RMSE of an unbiased estimator is the square
root of the variance, this is known as the standard error. For comparisons between
two things that may vary, and where neither are accepted as 'standard', the RMSE
is the square root of the average of the sum of the squares of the differences between
the two values, as shown in equation 3.2. RMSE is useful because it returns the
precision of the estimator in the same units used by the estimator[14]. For example,
a tracking estimator predicting an object’s velocity in meters per second will have a
RMSE measured in meters per second.

\[
RMSE(\theta) = \sqrt{MSE(\theta)} = \sqrt{E((\hat{\theta} - \theta)^2)} \quad (3.1)
\]

\[
RMSE(\theta_1, \theta_2) = \sqrt{E((\theta_1 - \theta_2)^2)} = \sqrt{\frac{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2}{n}} \quad (3.2)
\]

Another method of evaluation uses measures called recall which is the percentage
a detected object is overlapped by the related estimate and precision which is the
percentage an estimate overlaps its related object[9]. In this method a high recall does
not imply a high precision, nor the reverse. Precision and recall are more generally
defined as the ratios of the number of correct correspondences to the number of
established correspondences and the number of correct correspondences to the number
of actual correspondences[8].

A coverage test is used to determine whether or not an estimate is tracking an
object and if an object is being tracked. This coverage test is considered true if both
recall and precision are high. This is achieved by calculating the F-measure of the
recall and precision, and comparing it to a threshold value. The F-measure, also
called the $F_1$-score, is a statistical measure of a test's accuracy. The traditional F-measure is the harmonic mean of precision and recall and can be interpreted as their weighted average. The worst possible F-measure of a test is 0 and the best possible F-measure is 1. The traditional formula for the F-measure is:

$$F = 2 \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$  \hspace{1cm} (3.3)

Two identification error measures are also defined in this method, Falsely Identified Tracker (FIT) where an estimate trajectory is matched with the trajectory of the wrong object, and Falsely Identified Object (FIO) where an estimated trajectory switches to a different object. This can occur in cases of occlusion or when two objects pass close by one another. The fewer FIT and FIO occurrences, the better the tracking software.
Chapter 4

The Thesis

From the review of the state of the art it becomes clear that there are limits to the research that has already been done in the areas of object tracking and classification. While the underlying control theory concepts such as filters are well understood, most implementations of trackers start from those basic parts and build a solution unique to the particular problem faced by the developers. This provides a solution that is tightly coupled to the problem, but leads to a duplication of work. Also, because most tracking implementations are specific to particular problems, comparing their relative performances becomes difficult.

4.1 Statement of Thesis

By constructing a tracking framework which allows an easy switching of obstacle prediction and matching methods a better understanding of the relative merits of the various prediction and matching methods can be achieved. The performance of the various tracking methods can be quantified and compared to each other. A simulation tool can be used to test various tracking methods against conditions and behaviours the target platform is expected to encounter. From these tests the tracking method that best meets the requirements of target platform can be selected.
4.2 Analysis

The GeoSurv II UAV project, which is the impetus for this work, requires an obstacle tracking system for its autonomy software. However, the computer hardware to run the autonomy software is still unspecified beyond weight and power restrictions, and other aspects of the GeoSurv II UAV's systems are in the process of being specified and developed. This includes the sensor suite and the associated obstacle detector.

To advance the GeoSurv II autonomy software a framework and tools for testing and evaluating potential tracking methods needs to be developed. This framework would allow the engineering teams to determine what tracking methods are best suited to the kinds of obstacles GeoSurv II will encounter and provide information on the computational requirements of the tracking methods to help firm up the specifications for the computing hardware.

This thesis shows such an obstacle tracking framework. It is designed to be somewhat independent of the sensor(s) and obstacle detector as those are being developed in parallel and full information on them was not available at the time of this writing. Therefore the only information the obstacle detector must absolutely provide to the framework is obstacle location. The capacity to include additional information from the obstacle detector should be available, but not necessary. Such an ability for customization increases the general usefulness of the tracking system framework, but the framework should not be dependent on the additional information.

By testing and comparing different prediction and matching methods system designers can select those methods which best suit their particular problem. However, as discussed in section 3.1 there are no standard data sets for evaluation of tracking systems. The development of such a standard is complicated by the fact that different tracking systems will have different requirements. Some applications, especially military ones, require few occurrences of FIT errors so that the designer can be certain the correct obstacles are being tracked. Other applications are more tolerant of FIT errors as they are only concerned with the presence of obstacles and not their identities.

The confidence that can be placed in the output of a tracking system or a classification system is important information for whatever uses the tracking system's
output. As discussed in section 3.4 some systems do report confidence values showing how much trust can be placed on their output. However, the methods for computing this confidence are not generally discussed.

4.3 The Research

The research in support of this thesis is focused on the development an obstacle tracking framework. This obstacle tracking framework serves as a testbench to evaluate obstacle prediction and matching algorithms. The result of this work is the OTC system. The OTC system provides a testbed for candidate tracking systems for the GeoSurv II UAV.

To demonstrate the OTC system and the tracking system evaluation, one obstacle matching method and two predicting methods have been implemented. The matching and predicting methods used were not chosen with the intent that they would be the methods used by the GeoSurv II UAV. They were chosen to demonstrate the implementation of different tracking methods and their evaluation.

Additionally, a scripted simulator has been developed to exercise the testbench. This was done for three reasons. First, for safety reasons. This was the OTC system’s first use and installing it untested onto an actual mobile sensor platform would have posed a risk both to the platform and anyone in the platform’s area of operation. Second, platform availability. At the time of the OTC system’s design and implementation there were no easily available platforms to run it on. The GeoSurv II UAV’s airframe and avionics were not complete and the sensor package and obstacle detector had not been finalized. Third, control and repeatability. Using a simulator and scripts allows the tester to control exactly what sort of obstacles the OTC system encounters and ensures that tests are repeatable. This allows the tester to ensure that the particular matching or predicting method they are looking at behaves in the same manner every time it receives the same inputs.

The development and testing of actual candidate matching and predicting methods for the GeoSurv II UAV is an important area of work for that project, but is outside the scope of this thesis. This also applies to integrating the OTC system with GeoSurv
CHAPTER 4. THE THESIS

4.4. CONTRIBUTIONS

II's autonomy software, both the existing components and those under development. Care has been taken, however; to simplify this integration.

4.4 Contributions

This research makes two contributions to engineering knowledge. It demonstrates that an obstacle tracking framework can be developed and it presents a method for indicating confidence in reports of an obstacle's state.

The confidence that can be placed in the output of a system is important information for anything that is making use of that output. Knowing this information allows the receiving systems to weight the output when making decisions using the output. This leads to better decisions being made. Presenting a method for establishing the confidence that can be placed in an obstacle tracker's output allows future works to implement this method or build on it.

An obstacle tracking framework is a useful addition to the state of the art for several reasons. First, it means that an off the shelf tracking system can be developed thereby reducing the difficulty of designing and constructing autonomous vehicles. Second, it shows that a testbench to evaluate different tracking methods and algorithms can be developed.

The OTC system advances the GeoSurv II UAV's development in three ways. First, it provides a testbed to evaluate various potential tracking methods. This will allow the GeoSurv II team to determine which tracking method to use. Second, once a tracking method is decided upon and integrated into the OTC system, the OTC system can act as the tracking system for the GeoSurv II UAV. The OTC system is designed to use the same communication method and software that the other GeoSurv II software does, as outlined in Sean Bassett's work[1]. Finally, the OTC system's measurement capabilities provide information that will assist in determining the required capabilities of the UAV's computing hardware, and in determining the schedulability of the software running on the system.

As the GeoSurv II UAV's systems are finalized and an appropriate tracking method selected, the OTC system can be used as the tracking system for GeoSurv II.
The framework serves as a working tracking system, and the instrumentation code used in collecting information for the tests is designed to be easily removed from the final, compiled tracking system by setting a compiler flag.
Chapter 5

OTC System Design

The Obstacle Tracking and Classification system (OTC system) was developed to accomplish the goal of creating a testbed for evaluating different tracking and prediction methods. This section discusses and describes the design of the OTC system.

The OTC system exists as part of a larger environment consisting of an obstacle detector, a platform state monitor and a path planner. The OTC system communicates with this environment using a publish/subscribe middleware to subscribe to two message types and publish two other message types. The subscribed messages are obstacle and platform state data messages, the published messages are error messages and tracking data from the OTC system.

Internally the OTC system is made from six separate components: two data receivers, an error publisher, a tracking data publisher, a tracker and a prediction component. The tracking and prediction components are the most complex, and their implementation requires and examination.

Additionally, as part of the published tracking data the OTC system computes a self assessment of the reliability of the tracking data, which is presented as a confidence value. The rational behind this confidence value and the formula for its computation are examined.

Finally, the OTC system has a number of configurable parameters controlling its behaviour. These parameters, while mentioned in earlier parts of this section are listed and given short explanations at the end of the section.
5.1 Operating Environment

In the OTC system, the tracker is separate from the obstacle detection system. This has the benefit of not restricting the choice of sensor suites for the tracker, at the cost of potentially adding additional communications between the obstacle detector and the tracking system. This may add some redundancy to the system. The OTC system was developed in this manner because the GeoSurv II UAV project, the impetus for the development of the OTC system, had not finalized the sensor or obstacle detector design when it became necessary to start work on the tracking system.

From the point of view of the OTC system, the environment in which it operates consists of four parts, as seen in figure 5.1. The obstacle detector, the platform state systems provide information to the OTC system which in turn provides information to the path planner. Both the OTC system and the path planner fall under the umbrella of autonomy systems.

![Figure 5.1: Mobile sensor platform control environment](image)

The platform state of the GeoSurv II UAV is currently designed to be supplied by the aircraft’s autopilot. The MicroPilot 2128 heli that is under consideration provides GPS localization data, airspeed and altitude measurements and pitch, roll and yaw sensors. This information is needed by the OTC system to map relative-
to-platform measurements to relative-to-universal-coordinate-system measurements. For the GeoSurv II UAV the universal coordinate system chosen is latitude-longitude using the IERS reference meridian[24] and meters above sea level for altitude. The IERS meridian is used because it is the meridian used by the GPS[25, 24]. For reference the IERS reference meridian is located roughly 5.31 arc seconds east of the Greenwich prime meridian[24].

5.2 OTC Interface

The OTC system interfaces with other systems using a publish/subscribe middleware. As a result, the OTC system does not need to know about the systems providing it with information, only that it is being provided. It also means that the OTC system needs to know nothing of the systems receiving the information it generates.

The existing GeoSurv II UAV autonomy software uses the Inter-Process Communication (IPC) middleware from the Carnegie Mellon University School of Computer Science / Robotics Institute[22] as described in section 2.7. The IPC configuration used by the OTC system uses centralized routing for messages. Centralized routing provides facilities for message logging and analysis. A non-centralized routing configuration could be used, and would not substantially affect the performance of the OTC system, but would not provide built in logging capabilities.

In the OTC system the IPC specific function calls and message handlers are, as much as possible, hidden behind local OTC function calls. This is so that if the middleware is changed the switch over should be as simple as possible for the maintainers of the OTC system, and require a minimum of changes to the code.

Finally, as seen in figure 5.2, there are four messages in the OTC system’s interface. There are two subscriptions, through which the OTC system receives obstacle observations and platform state information. And there are two publications through which the OTC system reports system errors and obstacle tracking information.

All four messages have a timestamp so that their age and relevance can be determined. A message that is too old is no longer relevant to the control of the platform and should either be ignored or given less weight than newer messages when decisions
are made. The timestamp format is hour:minute:second:millisecond.

The following three sections describe the units the OTC system works with and the goes into detail about the four messages the OTC system uses.

### 5.2.1 Measurement Units

The OTC system deals with a number of measured quantities: bounding box sizes, ranges, angles, locations, altitudes and times. These quantities require units to be intelligible and useful.

Meters are used for altitude, obstacle sizes and ranges. Meters were chosen for largely cultural reasons, and because they seemed to be the appropriate scale for the work being done.

Time is currently measured with millisecond accuracy, although it is stored as hours, minutes, seconds and milliseconds. This allows stored times to be dealt with as either time elapsed (mission clock) or time of day.

Angles in the OTC system are measured in degrees as opposed to radians. All angles that are relative to the platform are measured using a vector through the nose of the vehicle as zero degrees. Horizontally ninety degrees is to the clockwise of zero and vertically, ninety degrees is ‘up’ relative to the platform.

Locations of obstacles and the UAV are recorded using the global coordinate system of degrees of latitude and longitude, with altitude in meters above sea level.
5.2. OTC INTERFACE

Conversions between degrees latitude and longitude and meters are done using the World Geodetic System reference ellipsoid, specifically the one from the latest revision WGS84. This is the reference coordinate system used by the GPS[25]. The main reason WGS84 was chosen is because the GeoSurv II platform will use GPS and using the same reference simplifies operations. WGS84 is a widely used standard so the use of this reference ellipsoid should not be detrimental to the OTC system’s compatibility with other platforms.

5.2.2 Received Data

The OTC system requires certain information in order to perform its tasks. It requires information on the state of the platform and it requires information on the obstacles it is supposed to be tracking. Obstacle data is assumed to be measured relative to the platform, therefore; the platform state is needed to convert between the platform relative measurements and global coordinates.

Platform State

Because the OTC system deals with all of its tracking information in real world coordinates using Latitude, Longitude and Altitude ASL in meters, it needs to know where the sensor platform is in the same coordinate system.

The OTC system requires relatively accurate localization information (like that from a GPS receiver). It is assumed that the system publishing the platform state information deals with any necessary sensor fusion or filtering. This is not an unreasonable assumption for an autonomous vehicle as in addition to the OTC system, other systems have a reasonable need for information on platform state and it is more efficient to filter once, than replicate the filtering in several places. For the GeoSurv II UAV, such other systems potentially include the path planner and mission data logger.

The OTC system must receive the following state information from the platform:
• Latitude (degrees).
• Longitude (degrees).
• Altitude (in meters ASL).
• Pitch (degrees).
• Roll (degrees).
• Yaw (degrees).
• Speed (air or ground depending on the platform).
• Timestamp.

Obstacle Data

The OTC system requires information about the obstacles it is tracking, otherwise it cannot track them. All the information it receives is relative to the sensor platform, hence the requirement for the platform state as described above. The only assumption made by the OTC system about how the obstacle data is gathered, or what sensors and algorithms are involved, is that the measurements are all made relative to the platform.

Each obstacle’s information is not passed separately. Whenever the detector has finished a detection cycle (e.g. finished examining a camera image, or one complete radar sweep) it passes the information on all obstacles it has detected as a single data structure, including the number of obstacles detected.

New measurements are assumed to arrive from the obstacle detector at a known rate. This rate is used by the OTC system to determine how often it must perform predictions and how far into the future to predict. The matching of new observations to known obstacles is predicated on the rate being known. If the rate is not known, the predictions will not be accurate.

The OTC system must receive the following information on each obstacle the detector system detects:
CHAPTER 5. OTC SYSTEM DESIGN

5.2. OTC INTERFACE

- Range (meters).
- Horizontal Bearing (degrees).
- Vertical Bearing (degrees).
- Bounding Box width (meters).
- Bounding Box height (meters).
- Timestamp.

The range is the distance, in meters, of an obstacle from the center of the platform. The bounding box is the smallest box that could be drawn around the obstacle, completely containing the obstacle. It is used to describe the rough size of the obstacle and is measured in meters. The bounding box is not currently used by the OTC system, it is part of the received data because it could be used by the matcher, depending on implementation (section 5.3.1), and is potentially useful information for the autonomy software especially the obstacle avoidance routines. The bounding box is very dependent on the sensors being used. A camera system can provide a reasonable two dimensional bounding box, while most lidar systems would only be able to measure the obstacle in one dimension.

The horizontal and vertical bearings measure where an obstacle is relative to the platform. If, for example, an obstacle has a horizontal bearing of 90 degrees and a vertical bearing of 20 degrees, it will be located to the right of the platform and up 20 degrees.

Noise in Measurements

The obstacle data in the observations is converted from platform relative coordinates to the latitude-longitude coordinate system. Because the platform relative coordinate system is spherical and the latitude-longitude system is not the conversion between the two affects the noise present in the measurements.

The conversion is performed as follows: First, the platform’s orientation (pitch, roll, yaw) is combined with the obstacle data’s horizontal and vertical bearings to
determine the horizontal bearing from North, and the vertical bearing from horizontal. Once these bearings are known the angles are fed through trigonometric equations\(^1\) to determine the distance between the obstacle and the platform in the latitude, longitude and vertical axes. Figure 5.3 shows the basic concept.

![Figure 5.3: Coordinate conversion](image)

This transformation is non-linear and therefore will affect the noise in the measurements. For example, noise which was initially Gaussian will not be Gaussian after the transformation. An implemented predictor should take this effect into account. If it does not the results of the prediction will be biased or incorrect.

5.2.3 Published Data

The OTC system publishes two messages. It publishes error message when problems occur to assist in determining why those problems occur and fixing the causes. It also publishes information about the obstacles it is tracking. The production of this tracking information is the entire purpose of the OTC system.

\[^1\]The exact equations depend on where the obstacle is. For example, if the obstacle is in the same plane as the platform and is located between 0 and 90 degrees from north, the distance in latitude is \(\text{Latitude} = \text{Range} \times \cos(\theta)\), but if the obstacle is between 90 and 180 degrees from north, \(\text{Latitude} = \text{Range} \times \sin(\theta)\)
CHAPTER 5. OTC SYSTEM DESIGN

5.2. OTC INTERFACE

Error Messages

The OTC system generates error messages whenever it detects that something has gone wrong with its own operation. These messages are published to help the operator of the platform using the OTC system determine what is happening if a problem occurs.

The error message format used by the OTC system is intended to be the standard format for the GeoSurv II's systems and has four parts: a system id, an error number, an error string and a timestamp. The system id is a number that identifies the system reporting the error. The error number is a number that identifies the error that has occurred. The error string is a short, human readable explanation of the error. The timestamp shows when the error occurred. The error number and string serve the same purpose, to identify the error. The string is intended to be logged or displayed for the human operator of the system, while the number is intended for the use of software systems responding to the error.

Tracking Data

The OTC system publishes a sub-set of the information it retains for each obstacle. Some of the information it maintains is needed only for internal use, and therefore is not published.

The publisher collects and sends the location, bounding box, airspeed, the classification and the confidence value (section 5.4) of each obstacle along with a timestamp. As with obstacle observations from the obstacle detector the data for all the obstacles known to the OTC system is published as a set.

Published information is normally drawn from an obstacle's predicted state. However, in the case that the publisher is run between the detection of a new obstacle and the first run of the prediction routines on that obstacle, the measured data is used by the publisher. In this case, the confidence value associated with the new obstacle is low.
5.3 OTC Internal Structure

Before covering the behaviour of each part of the OTC system, it is necessary to
describe the state information maintained by the OTC system. There are two sets
of state information, the platform state and the known obstacles list. The platform
state consists of the latest information provided by the platform state messages. The
content of these messages, and therefore the stored platform state, is detailed in
section 5.2.2.

The known obstacles list is a list of all the obstacles that have been detected by
the obstacle detector system up to the current time. The data stored by the list
consists of the most recent observation of the obstacle, the most recent prediction
of the obstacle's state (including location), the obstacle’s classification and the fuzzy
variables used to make the classification. The fuzzy variables are discussed in depth
in section 5.3.2 along with the classification method.

Internally, the OTC system is divided into six independent threads as seen in
figure 5.4. The role of each thread is described below. Communication among threads
is accomplished through a combination of signals, semaphores and shared memory.
Figure 5.4 also shows the publish/subscribe middleware connecting the OTC system
and external systems. The large block labelled OTC System is an expansion of the
OTC System block in figure 5.1.

Figure 5.4: Block diagram of OTC system threads

The obstacle data receiver subscribes to the obstacle observations published by the
obstacle detector. The observations are described in section 5.2.2. If no publications are received within a programmed amount of time, an error is signalled to the error publisher. If, however a publication is received, the data is copied to the appropriate location in shared memory and signals are sent to the tracker.

The period of obstacle detector messages must be known before the OTC system is run, and is configurable. This period has two uses. First, it can be used to determine if the obstacle detector is publishing messages at all. Second, the period value is needed by the tracker and the prediction thread. The prediction thread needs to know how far into the future to make its predictions so that the tracker thread can correctly match predictions to new observations.

The platform state receiver behaves similarly to the obstacle data receiver, except it subscribes to platform state messages, as described in section 5.2.2. The platform state receiver also waits for a set amount of time and will report an error if no platform state is received within that time. The time is a configurable parameter of the system and can be set using the OTC configuration file described in section 5.5.

The tracker thread is a wrapper around a black box known as the matcher. The tracker thread receives obstacle observations from the obstacle data receiver and platform state from the platform state receiver. It converts the obstacle data from platform relative coordinates to global coordinates and then passes the obstacle observations and the known obstacles list to the matcher.

The matcher compares observations from the obstacle detector to the known obstacle list and takes one of two actions. If an observation matches a known obstacle the matcher updates the obstacle's most recent observation with the new data. If an observation does not match a known obstacle the matcher creates a new entry in the known obstacles list using the observation data. The matcher is treated as a black box because there are many potential methods for performing matching. Treating the matcher as a black box with defined inputs and outputs allows different matchers to be swapped in for testing. Details of the implemented matcher are given in section 5.3.1.

The prediction thread waits for a signal from the tracker to indicate that new obstacle data is available. This is a timed wait and the prediction thread will also run if the time limit is exceeded. The time limit is included because the tracker
needs up to date predictions to function correctly. In the event that none of the known obstacles are currently being observed the timer ensures the predictions of their states are still updated.

When either new observations are available or the time limit expires, the prediction thread steps through the known obstacles list and runs each obstacle through the predictor to obtain a new estimate of the obstacle’s state. The thread also classifies the obstacles. The predictor is, like the matcher, a black box with defined inputs and outputs that can be switched easily. Details on the two predictors implemented for the testing in section 7 are covered in section 5.3.2.

The black box nature of the predictor and matcher to the rest of the OTC system is what allows the OTC system to be used as a framework for testing various matching and predicting algorithms and methods.

The tracking publisher thread waits on its own periodic timer, and once every publishing period it publishes the tracking data of all the obstacles in the known obstacles list. This tracking data is covered in section 5.2.3.

The error publisher thread simply waits for an error signal from one of the other threads and publishes the error passed to it when such a signal is received.

The tracking publisher and the error publisher are different threads because they publish different messages and the publication is triggered by different conditions. The tracking publisher by a periodic timer, the error publisher by asynchronous error signals.

The OTC system includes a compile switch to insert instrumentation code to allow time and memory usage measurements to be taken. To build a version of the OTC system for final installation on a mobile sensor platform, this instrumentation code should be removed by recompiling the OTC system without the TIMELOG switches present in the Makefile. The instrumentation code produces a set of log files which can be examined after the OTC system has run. These logs and the other methods of evaluation allow the effectiveness of various types of prediction methods to be determined.
5.3.1 Tracking and Matching

The tracker thread interacts with the prediction thread in three ways. It provides new data for known obstacles, determines if new obstacles have been found and receives predicted obstacle state from the prediction thread.

In order to ensure that the predicted obstacle states are useful to the matcher, the period of the observation messages must be known. For example, if the period of observation messages is 5 time units and an observation message with a set of observations arrives at time T, the next message will arrive at time T+5. Therefore, a prediction made about obstacles using data from time T must be a prediction about the obstacle's state at time T+5, or the tracker will not be able to correctly match the time T+5 observations to the predictions. The observation message period should be set in the OTC system configuration file (see section 5.5) by the end user.

As discussed above the tracker thread of the OTC system uses a black box called the matcher to compare each obstacle in a new set of observations to the list of known obstacles. The matcher is defined as a function within the OTC code. Any function that uses the correct prototype and returns the correct values can be used as a matcher provided it is compiled into the object file (otc_matcher.o).

The prototype for the function is:

```c
int otcMatchingFunction( otc_tracking_info *tracked_obsts,
                          int maxTrackedObstacles, sf_contact_list measurements)
```

The matching function returns the success, or error, as an integer and takes the list of known obstacles (tracked_obsts) and the list of new observations (measurements) as parameters. Information on matches is passed out through the known obstacle list.

If an observation of an obstacle matches a known obstacle, the matcher reports that they are the same obstacle and saves the observation data as part of that obstacle's entry in the known obstacles list. If an observation does not match any known obstacle the matcher determines that it is a new obstacle and adds a new entry to the list of known obstacles.

The criteria for determining what constitutes a match are dependent on the matching algorithm implemented.
CHAPTER 5. OTC SYSTEM DESIGN  5.3. OTC INTERNAL STRUCTURE

Implemented Matcher

The implemented matcher used for the tests described in section 7 is a position based matcher. If a known obstacle and an observation are close enough to each other, position wise, they are considered to be the same obstacle.

What close enough to match is, is determined by a configurable parameter of the OTC system known as the matching radius. The matching radius has three components: latitude, longitude and altitude. The matching radii can be adjusted to fit the requirements of the platform running the OTC system.

If no known obstacles match an observation, the matcher assumes that a new obstacle has been detected and adds it to the list of known obstacles. The observation data is used as the new obstacle’s starting position.

This method is naive in that it accepts the first found match rather than examining all possibilities and selecting the best match. This can have effects when two or more obstacles pass close to each other, see section 7.3.6. Depending on the order of the obstacles in the known obstacle list, the observations could be matched to the wrong entry in the known obstacle list.

Information on the relative size of the obstacles being matched is not considered. At the time of writing it was not certain that the bounding box information (see section 5.2.2) would be accurate enough to be of use. The effectiveness of adding this capability to the matcher is an open question for future work.

5.3.2 Prediction and Classification

Every time there are new observations, or if the predict period is exceeded, new predictions are made. The predictor function is called once for each obstacle in the known obstacles list when this happens. The cycling through of all the known obstacles and predicting their locations is known as the predict loop.

The OTC system implements a prediction routine as a function called by the Prediction thread. This function has a specific prototype:

```c
void otc_filter(otc_tracking_info *obst, otc_location_info measured, double d_lat_m, double d_lon_m, int new_measurements,
```
The implementation of the function is a black box to the Prediction thread and the rest of the OTC system. The function's parameters are the obstacle being predicted (obst), the measurements (measured), the sizes of degrees latitude and degrees longitude in meters (d_lat_m), (d_lon_m) for use if the predictor needs to convert between the two systems\(^2\), a flag to indicate if there are actually new observations or if the predictions are for an obstacle that is not currently observed (new_measurements) and a pointer to pass out the results (results). d_lat_m and d_lon_m are included because these values change depending on how far north or south a position is. This means they should not be hardwired into the prediction function.

To implement a filter or other prediction routine, a programmer must create a function that matches this prototype and compile it to an object file (otc_filter.o).

In addition to the two implemented prediction functions discussed in section 5.3.2 a skeleton prediction function file is provided with the OTC system to simplify the development of future predictors.

Known obstacles are classified as stationary, dynamic or unknown. A stationary, or static object is one that does not move. For example a house or radio tower. A mobile, dynamic object is one that does move such as an aircraft, bird or windblown debris. Finally there are unknown objects, those for which there is not yet sufficient information to make a classification decision.

Two variables are maintained to determine an object's classification\(^3\). They are fuzzy values used to determine if the obstacle should be considered static or mobile. The possible values held by these two variables are in the range from 0 to 1. The variables are the static fuzzy class value and the mobile fuzzy class value. The initial values of the two variables are a configurable parameter.

\(^2\)A constant conversion factor is not used because the number of meters in a degree is dependent on location.

\(^3\)There are two variables to simplify the presentation of the confidence value formula in section 5.4. A single fuzzy value could be used, increased as the obstacle is determined to be mobile, decreased as the obstacle is determined to be stationary. This would produce the same result but would not substantially reduce the work required for classification or calculating the confidence value.
The fuzzy class values are adjusted in tandem as more information is gathered, as one is increased, the other is decreased. Every time through the prediction loop each object's predicted airspeed is compared to the OTC system's airspeed threshold. If the predicted airspeed is higher than the airspeed threshold, the obstacle is assumed to be mobile. If the predicted airspeed is lower than the airspeed threshold, the obstacle is assumed to be static. The airspeed threshold value is one of the OTC system's configurable parameters (see section 5.5). It is not hardwired to zero as it is assumed that noise and measurement error can result in an obstacle having the appearance of a low airspeed.

Once an obstacle's predicted airspeed is compared to the airspeed threshold, the two fuzzy values are adjusted. If the comparison indicated that the obstacle was likely moving, the mobile fuzzy class value is increased and the static fuzzy class value is decreased. If the comparison indicated that the obstacle was likely to be stationary, the static fuzzy class value is increased and the mobile fuzzy class value is decreased. The increment and decrement sizes are the same in all cases, and the setup of the OTC system is such that the sum of both fuzzy value variables is always 1.0. The increment and decrement sizes are adjustable in the OTC system configuration but cannot be larger than 0.5.

The reported classification of an object is determined by comparing the difference between the two fuzzy classification variables to a threshold. If their difference exceeds the threshold then the larger of the two is reported as the object's classification. For example, if the static fuzzy class value is 0.8, the dynamic fuzzy class value is 0.2 and the threshold is 0.4, then the difference between the two is 0.6 which exceeds the threshold and the obstacle will be reported as being stationary. If the difference between the two fuzzy variables does not exceed the threshold the obstacle is reported as being in the unknown class. The classification threshold is also an adjustable setting of the OTC system configuration.

**Implemented Predictors**

A KF based predictor and an EKF based predictor were implemented to demonstrate the OTC system and show how two predictors can be compared to one another. A
KF was chosen because the KF is one of the simplest and easiest filters to implement. The EKF was chosen as a more complex filter to demonstrate the performance tests. The similarity of the predictors also helped serve as a sanity check. Failure to produce similar results when passed the same data would have been a clear sign that there were mistakes in the OTC system software.

Neither predictor is intended to be the final predictor used by the GeoSurv II system. They were chosen to demonstrate how the OTC system can be used as a testbed for predictor comparisons.

The KF predictor is implemented using the obstacle’s velocity, rate of climb or decent, heading and angle of climb as the state variables. All these values were assumed to be constant resulting in linear motion of the obstacle. This assumption is not necessary for a KF, and the assumption was made for ease of implementation. The final filter, or other estimator used by an eventual target OTC system may make other assumptions. It is not an unreasonable assumption, and successful vehicle trackers have made the assumptions of constant acceleration[19] or constant velocity.

The states used by this filter implementation are the location of the obstacle in latitude and longitude and altitude, the velocities of the obstacle in those directions, including rate of climb (or decent), and the heading and angle of climb (or decent). The obstacle state vector \( \hat{x}(t_k) \) as described in section 2.5.1 is shown in equation 5.1

\[
\hat{x}(t_k) = \begin{bmatrix}
lat(t_k) \\
lat(t_k) \\
lon(t_k) \\
lon(t_k) \\
alt(t_k) \\
alt(t_k) \\
head(t_k) \\
angle(t_k)
\end{bmatrix}
\]

\( \hat{x}(t_k) \) is the heading of the obstacle, using north as 0 degrees. \( \angle(t_k) \) is the angle of climb, or descent, of the obstacle. Both are filtered because they come from noisy measurements, and both are used by the classifier to determine the dimensionless airspeed of the obstacle.
The obstacles are assumed to be moving in a straight line at a constant velocity. This is not likely to be an accurate model of several potential types of obstacles that the GeoSurv II could encounter, birds for example. However, it is sufficient to demonstrate that the predictors work, and that the OTC system can be used as a testbench for evaluating prediction methods.

The discrete KF equations, equations 2.1 and 2.2, contain the terms

\[ x_k = F_k x_{k-1} + w_k \]  \hspace{1cm} (5.2)

\[ z_k = H_k x_k + v_k \]  \hspace{1cm} (5.3)

Where

\[ F_k = \begin{bmatrix}
1 & \delta t & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \delta t & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \delta t & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \]  \hspace{1cm} (5.4)

\[ H_k = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \]  \hspace{1cm} (5.5)

where \( \delta t \) is the, constant, time between observations, and

The result of the first step of the Predict phase, the predicted \textit{a priori} state,
equation 2.3, is therefore;

$$
\begin{bmatrix}
lat_{k-1|k-1} + \dot{lat}_{k-1|k-1}\delta t \\
\dot{lat}_{k-1|k-1} \\
lon_{k-1|k-1} + \dot{lon}_{k-1|k-1}\delta t \\
\dot{lon}_{k-1|k-1} \\
alt_{k-1|k-1} + \dot{alt}_{k-1|k-1}\delta t \\
\dot{alt}_{k-1|k-1} \\
head_{k-1|k-1} \\
angle_{k-1|k-1}
\end{bmatrix}
$$

The initial states in the state matrix, \( x_{0|0} \), for an obstacle are taken from the first observation made of that obstacle. The initial covariance matrix, \( \Phi_{0|0} \) is given an initial value, \( Pcoeff \) an adjustable parameter of the predictor, as shown in equation 5.7. The tests in Section 7 use a \( Pcoeff \) of 1.

$$
\Phi_{0|0} = I_8 \ast Pcoeff 
$$

The state space and assumptions for the EKF predictor implementation are the same as to those used by the KF. This means that the predictive performance of the EKF will be identical to the KF when the motion of the obstacle being tracked behaves in a fashion that can be described as linear. However, the EKF based predictor performs linearizations in all circumstances which means that the EKF should require, on average, more processor cycles than the KF. These linearizations are the calculations of the state transition matrix Jacobian and the observation sensitivity matrix Jacobian.

The Jacobians are calculated through complex-step differentiation. Complex-step differentiation is a method for calculating a numerical approximation of the first derivative of a function that avoids the round-off error common in many standard methods[26]. The basic equation for determining the derivative of a function \( f(x) \) using complex-step differentiation is shown in equation 5.8. In equation 5.8, \( j \) is the imaginary unit and \( h \) is a small step size. In the EKF predictor’s implementation \( h \) is the floating point epsilon of the computer running the OTC system, the difference between 1 and the least value greater than 1 that is representable.
The initial attempt at implementing the EKF based predictor involved an adaptive EKF [5] which made fewer assumptions about the obstacle model and could be expected to handle obstacles moving in a non-linear fashion much more accurately than the KF based filter. However, difficulties encountered in implementation and time restrictions forced a change. The adaptive EKF was simplified to use the same linear obstacle model as the KF based predictor, which mean that the EKF’s accuracy and noise sensitivity should be identical to the KF’s. Because the EKF and KF now use an identical obstacle model, and that model is linear, the Jacobians for the EKF are constant. However, the Jacobian calculations are still performed by the EKF so that the overhead they impose may be examined.

\[
\frac{\delta f}{\delta x} \approx \frac{Im[f(x + jh)]}{h} \quad (5.8)
\]

5.4 Confidence

The purpose of the filters and prediction methods used by the OTC system is to minimize uncertainty and noise in the data produced so that the best information is available for decision making. However, completely eliminating uncertainty is not possible. For this reason the tracking and classification system produces a confidence value for each obstacle it tracks.

The confidence value is a measure of how reliable the OTC system believes the data it is reporting is, and is intended to allow the system receiving the tracking system’s output to weight that output when making a decision. It is a self assessment of the tracking system’s performance. The confidence value has a range of 0 to 1. 1 is complete confidence and 0 is a complete lack of confidence. Thus a high confidence value means that, according to the tracking system’s criteria, the obstacle data is probably very accurate and should be weighted accordingly. A low confidence value means the opposite, the tracking system considers the obstacle data to be currently less accurate than it is capable of producing.

The confidence values are based on three factors: time elapsed since the last observation, the classification value of the obstacle, and the number of observations.
that have been made of the obstacle.

Despite the best prediction methods, the longer it has been since an obstacle has been observed the more likely it is that error has crept into the predictions. Refining the prediction method and prediction model reduces this problem but it cannot be eliminated completely. As such the confidence value goes down as the time since the last observation increases.

An obstacle with many collected observations is less likely to be a false positive and is more likely to be accurately predicted given the greater feedback. Thus the more observations that have been made, the higher the confidence in the prediction.

Finally, the more certain the classification is, the more certain it is that tracking and predicting are being done correctly. If an obstacle has been classified as either stationary or dynamic the corresponding classification fuzzy value is included in the confidence value formula. Obstacles that are currently classed as UNKNOWN do not have either of the fuzzy values included in their confidence value calculation and are therefore obstacles that the OTC system is much less confident in.

The formulas for calculating the confidence value are:

\[
T_{ob} = \frac{P_{pred}}{N_{pred} \times P_{pred}} = \frac{1}{N_{pred}} \tag{5.9}
\]

\[
Confidence = \left[ T_{ob} + Class_{fv} + \frac{n}{N_{obs}} \right] \times \frac{1}{3} \tag{5.10}
\]

Where:

- \( P_{pred} \) - Length of the period between predictions, usually in msec.
- \( N_{pred} \) - Number of prediction periods since the last observation of an obstacle (has a minimum of 1).
- \( T_{ob} \) - One over the number of prediction cycles since last observation (has a maximum value of 1)
- \( Class_{fv} \) - The fuzzy value of the current classification (see below).
- \( N_{obs} \) - A sliding window of \( N_{obs observation sets received} \) (configurable parameter)
• \( n \) - The number of observations made, up to \( N_{obs} \).

All three factors produce values between zero and one on their own, so their sum is divided by three to normalize the confidence value.

The classification fuzzy value is chosen from the two fuzzy variables used by the classifier. If the obstacle is classified as stationary, \( \text{Class}_{fv} \) is set to the value of the stationary fuzzy variable. If the classification is mobile then \( \text{Class}_{fv} \) is set to the value of the mobile fuzzy variable. If the classification is unknown then \( \text{Class}_{fv} \) is set to zero. The OTC system is much less confident in the data it maintains for an unclassified obstacle than it is in the data for a classified one.

The OTC system’s confidence value can be further improved by incorporating the self assessment of the obstacle detector system providing the observations. If the obstacle detector provides a confidence value for its observations this term can be added to equation 5.10. It was not done for this version of the OTC system because, at the time this research was conducted, the GeoSurv II’s obstacle detector system was in the early stages of design and its ability for self assessment was not yet addressed.

The OTC system’s confidence value is central to its handling of false positive and false negative obstacle detection reports from the detector. The obstacle detector design should minimize the occurrences of false positives and false negatives, however; the OTC system cannot assume that the obstacle detector is capable of completely eliminating these errors. A false positive observation will generate a false obstacle in the known obstacles list, but the number of observations \((n\text{ out of } N_{obs})\) will be low and the false obstacle will not be observed often decreasing \( T_{ob} \) as well. This will result in a very low confidence value for the false obstacle.

False negatives in the detector will result in missing observations for known obstacles, giving those obstacles a lower confidence value as \( T_{ob} \) will be increased. However, tracking of those obstacles is still possible and they will be reacquired if the predictions are accurate enough. The OTC system relies on the detector to have an acceptably low number of false negative reports.
5.5 Configuration

The OTC system has a number of configurable parameters which have been mentioned in the earlier parts of this section. These parameters are used to adjust the behaviour of the OTC system without requiring it to be recompiled. The values for these parameters are stored in a text file, .otc_config, and are read at run time. Currently there are no default values. If any value is not set in the configuration file the OTC system will publish an error message and fail to run.

The fifteen tunable parameters are:

- Maximum number of tracked obstacles.
- Heartbeat Time, the period the receiving threads wait before sending an error about receiving no data (milliseconds).
- Publish Period, the time between OTC tracking data publications (milliseconds).
- Predict Period, the maximum time between predictions (milliseconds).
- Latitude Match Radius, used by positional matchers to determine how close a measurement must be to a predicted position to be considered a match (meters).
- Longitude Match Radius, the same for longitude (meters).
- Altitude Match Radius, the same again for altitude (meters).
- Default Class Value, the default fuzzy classification value.
- Minimum Class Change, the smallest change possible for the fuzzy class value.
- Maximum Classification Value, the highest possible fuzzy class value.
- Minimum Classification Value, the smallest possible fuzzy class value.
- Classification Threshold, the difference between the fuzzy class values required for a decision to be made.
• Classification Airspeed Threshold, how low the obstacle predicted airspeed must be for it to be considered stationary (meters per second).

• Maximum Observations for Confidence, the number of observations made of an obstacle required to have the highest confidence in its predictions.

• Internal External Age Difference Limit, the maximum allowable difference in age between platform state and obstacle observations before results drawn from their comparison can be considered invalid (milliseconds).
Chapter 6

Simulation

A simulator was written to test the OTC system. This was done for three reasons. First, so that the testers could be confident that the OTC system was ready for field testing. Second, because the target platform was not yet ready and third, because it allows the tests to be controlled and repeatable.

The simulator has six parts, a script, a tester, a publisher, a data receiver, an error receiver and a noise generator. Each part’s role is examined below, however; the noise generator is examined in more detail. The noise generator’s output is also analyzed to ensure the simulator’s output is correct when performing OTC system tests.

6.1 Simulator Software

The six parts of the OTC simulator are shown in figure 6.1 which also details the flow of information through the parts of the simulator. All parts of the simulator, except for the script, are independent programs that are invoked and run separately.

The scripts are ASCII text files which contain virtual platform state measurements and virtual obstacle detector observations with associated time stamps. Script files are used to ensure that tests can be repeated with exactly the same data each time through the test.

The script tester is used to verify the formatting and structure of a script, reporting
syntax errors before the script is run through the noise generator or the data publisher. It ensures that the messages published by the data publisher will be formed correctly and not cause problems with the OTC system. It does not, however, ensure that obstacle and platform behaviours described in the script are realistic.

The noise generator reads a tested script and creates a noisy version of that script, with the noise based on a parameter description file. The noisy version of the script is a new script file. The noise generation does not change or affect the original script. This allows repeatability of any tests run using the simulator. The noise generator is the most complex part of the system and will be examined more thoroughly in section 6.1.1.

The data publisher reads in a script and publishes messages based on the script. The data publisher simulates the obstacle detector and platform state monitor systems described in figures 5.1 and 5.4. During a simulation, all of the OTC system's input is generated by the data publisher.

The data receiver subscribes to the OTC system’s tracking data messages and logs them so that the OTC system’s performance can be reviewed. The data receiver takes the place of, or supplements, the path planner portion of the autonomy system.
as seen in figures 5.1 and 5.4.

Finally the error receiver subscribes to the OTC system’s error messages. It logs the error messages so that errors encountered by the OTC system can be reviewed. Errors may be deliberately invoked by scripts to test the OTC system’s ability to detect the error conditions, or errors may occur naturally and their causes need to be determined and corrected.

### 6.1.1 The Noise Generator

The noise generator is designed so that the original script is preserved, and that a new, noisy script is created and saved. This allows comparisons between the two files, and the OTC output they generate. It also allows the repetition of any test and ensures that the OTC system is displaying consistent behaviour. Another benefit of separating the noise generator and the scripted data publisher is that the amount of time required to generate the noise is not a factor in the running of the script.

The noise generator uses a parameter description file to determine what kinds of noise to add to a measurement. Any number of parameter description files may be created, to represent different noise conditions, but only one file is used per generation of a noisy script.

The parameter description file contains a list of each of the UAV state variables (section 5.2.2) and obstacle detector variables (section 5.2.2), along with an associated noise distribution type, indicated by a keyword, and the parameters required to generate a random value according to that distribution. For example, when generating random values with a normal distribution, the entry in the parameter description file contains the keyword `normal` and the desired mean and standard deviation ($\sigma$).

Each variable can be set individually, but cannot be changed once noise generation has started. All instances of any given variable will have the same noise settings while the noise generator is running. This means that if there are two obstacles in a script, they will both have the same noise parameters for each of their variables. For example, if a parameter description file sets the obstacle range variable to have normally distributed noise with a standard deviation of 0.5 meters, then all obstacle ranges in the target script will have this noise. In the current design it is not possible
6.2 Simulated Noise

Noisy script files are generated through the use of the Gnu Scientific Library (GSL)[27]. The current version of the noise generator supports three kinds of noise, no noise, uniformly distributed noise and normally distributed noise.

More kinds of noise can be added by making two modifications to the noise generator. First a keyword for the kind of noise must be defined and added to the probability parameter file parser. This keyword is programmer defined and describes the type of noise. Currently, the keywords are 'normal' and 'uniform' for normally and uniformly distributed noise, respectively. Second, The GSL function calls for the desired type of noise must then be added to the script parser/modifier section of the noise generator. This will ensure that the noise generator can modify scripts correctly.

The three current noise types were implemented for specific reasons. The normal distribution type was chosen because it is a common distribution in natural systems[14]. The uniform distribution was chosen as a simple test for adding a new distribution type to the generator. The no noise option was added so that noisy scripts could be generated that added noise to only a subset of the variables. This was intended to simplify various aspects of testing the OTC system.

6.3 Noise Validation

To validate noise generation, a script was created with five hundred platform state entries and five hundred obstacle state entries, both the obstacle and the platform were stationary. This script was run through the noise generator one thousand times producing a matching number of output scripts. This was done for each implemented noise distribution type.

The variables with the noise added were extracted from the output scripts, yielding 500,000 samples. These variables were read into MATLAB, their histograms plotted and their parameters evaluated. The number 500,000 was chosen as the version of
MATLAB available ran out of memory when larger numbers of samples were provided.

For a normal distribution with a mean of 45.45 degrees North with a standard deviation of 0.000044988 (which represents 5m in degrees of latitude at 45.45 degrees North) and with 500,000 samples, a roughly bell-curve shaped histogram was obtained as seen in figure 6.2. This curve matches what would be expected of a Gaussian function. The measured standard deviation was 0.0000448. The mean and median values of the graph are both 45.450001.

This compares very well with the input parameters. The measured mean and median agreed with the ideal values to five places past the decimal. The measured standard deviation is farther from the ideal with a 0.3 percent difference. However, this difference is small enough that the noise generator can still be considered accurate for the purposes of the OTC system tracking and predictor testing.

![Figure 6.2: Histogram of Gaussian noise with stdev 5m at 45.45 degrees north](image)

For a normal distribution with a mean of 75.7 degrees West with a standard deviation of 0.000063916 (which represents 5m in degrees of longitude at 45.45 degrees North) a bell shaped distribution was also obtained as seen in figure 6.3, with a mean of...
75.699996 and a median of 75.699997. The measured standard deviation is 0.0000635.

Again this compares well with the input parameters. The differences between the measured and ideal mean and median are less than 0.000006 percent and 0.000004 percent respectively. The measured standard deviation is 0.6 percent off of the ideal. Again this is accurate enough that the noise generator can be used with confidence for this distribution.

![Histogram of Gaussian noise with stdev 5m at 75.7 degrees west](image)

Figure 6.3: Histogram of Gaussian noise with stdev 5m at 75.7 degrees west

For the uniform distribution with a spread of -2 to +2 about 20, with 500,000 samples, a roughly flat histogram was obtained as seen in figure 6.4, with a maximum value of 21.999912 and a minimum value of 18.002577, around a mean of 19.981987 and a median of 19.961397.

The measured mean is 0.09 percent off of the ideal and the measured median is 0.2 percent off. The difference between the ideal spread and measured spread of the values is less than 0.07 percent. This is accurate enough for the purposes of the OTC system tests.

For the no noise option the histogram was a single spike with no deviation. Exactly what was expected if no noise was added. There was no change between the original value of the variables, and the values in the output scripts.

63
Figure 6.4: Histogram of uniform noise on range measurements of 20m±2m
Chapter 7

OTC Tests

The OTC system must be tested to show that it works, and to demonstrate how it can be used as a testbed for various matching and prediction algorithms. For this purpose, six metrics with associated tests have been defined: prediction accuracy, measurement convergence, coverage, noise sensitivity, time and computer resource usage, and finally a set of scenarios with various obstacle behaviours. Both of the predictors described in section 5.3.2 have been tested using these six tests and their relative performances compared. Each test is conducted using one or more scripts for the simulator discussed in section 6. These tests and the modular nature of the matcher and predictor systems serve to demonstrate how the OTC system can serve as a framework for evaluating tracking methods.

As the OTC system will encounter noisy environments when it operates in non-simulated environments, the tests have noise added to the simulation scripts. A standard level of noise is defined in section 7.1, and the reasoning for the choices made is outlined.

Additionally, an overview of the computer system used to run the tests is given.

7.1 Noise in the Tests

The noise parameters used to generate the noisy test scripts are based on either known values for a specific type of measurement and transducer or on the most reasonable
values that could be determined.

The values described here, and listed in table 7.1, are treated as the standard level of noise. This is the base level of noise referred to in the noise sensitivity tests in section 7.3.7. This is the standard for the tests described in this section and should be adjusted depending on the platform being considered.

The latitude and longitude were exposed to Gaussian noise with a standard deviation of 6.7m. This value was chosen to test both the general behaviour of the OTC system and the specific behaviour of the OTC system under the conditions expected for GeoSurv II UAV operations.

The remaining values were chosen because they were deemed large enough to cause the data to require filtering and small enough not to make the sensors appear unusable. The standard deviations used for the other platform state measurements were 2 meters for the altitude and 2.5 degrees each for the pitch, roll and yaw. The original intent was to match the behaviour of the internal sensors of the MicroPilot 2128 heli autopilot being considered for the GeoSurv II UAV, however; the required data was not available at the time of this writing.

The obstacle detector values all use Gaussian distributions. Their standard deviations were chosen somewhat arbitrarily as information on the behaviour of the sensors intended for the GeoSurv II UAV was not available at the time of testing. The standard deviations used are: 2 meters for the range, 2.5 degrees for the horizontal (H) and vertical (V) bearings, and 0.5 meters for the bounding boxes (BB).

7.2 Testing System

The tests were run on a computer system with a 2 GHz processor and 1 GB of RAM. The operating system was the Fedora Core 6 Linux distribution, with Linux Kernel 2.6.

The computer system used to run the tests of the OTC system imposed a number of restrictions on the tests. First, the system was not a real time system, meaning that measurements of response time have limited accuracy and are mostly useful for relative comparisons of predictors. Second, it was not possible to limit the system
to only running the OTC system, the middleware and the simulator software. This resulted in further inaccuracies in time measurements due to interrupts from other processes. Finally, the clock available on the system for the simulator was limited to 1 second accuracy. This limit on the simulator meant that the simulated obstacle detector could supply observations at a maximum rate of once every second.

Even with these limitations the tests described in this section demonstrate that the testing methodology works and that comparisons between predictors and matchers can be made.

### 7.3 Performance Measures and Tests

Tracking system evaluation measures are discussed in section 3.5. They include prediction accuracy, coverage and the handling of scenarios where observation is lost, and where obstacles are occluded. Additionally, the sensitivity of the OTC system output to noise and its consumption of resources such as RAM and CPU cycles are of interest.
7.3.1 Accuracy

Accuracy is the measure of how closely the predictions match reality, i.e. the difference between the OTC system output and the actual locations of obstacles being tracked. This measure is normally called forecast error.[14]

For accuracy analysis, the RMSE of the output of the OTC system from a noisy script and the base script used to produce it will be used. RMSE was chosen for this characterization of the OTC system’s performance because it is one of the standard error measures used in statistics.

Accuracy is tested using a base script with the platform moving in a straight line with an obstacle crossing it’s field of view at 45 degree angle. The obstacle moves in a north-westerly direction, crossing in front of the platform. The platform moves north, crossing the obstacle’s path behind the obstacle. A simulated collision between the obstacle and the platform is avoided by setting the initial positions and velocities such that the obstacle reaches and passes the point where the two trajectories intersect well before the platform does. Figure 7.1 shows the behaviour of the platform and obstacles. Note that the axis are not to scale, figure 7.1 serves only to illustrate the relative motion of the platform and obstacle. The base script had 50 obstacle observations and 50 platform state measurements.

The tests used twenty scripts that were created from the base accuracy test script. Ten of the scripts have no noise and ten have the standard amount of noise added. These twenty scripts are known as the accuracy test scripts.

The KF based predictor, when run against the accuracy test scripts with no noise, has a RMSE for the predicted location is 0.14m for latitude and 0.31m for longitude and 0m for altitude. The altitude measure is not unexpected as the obstacle in the scripts moves at a constant altitude. The error that occurs in the latitude and longitude is an artifact of the fact that the KF expects there to be some level of noise in its observations.

Figure 7.2 shows the predicted obstacle path from noiseless observations (dashed line) compared to the actual path of the obstacle (solid line).

With the addition of the standard noise (see section 7.1) the accuracy of the predicted locations decreased with the average RMSE for the latitude increasing to
CHAPTER 7. OTC TESTS  7.3. PERFORMANCE MEASURES AND TESTS

Figure 7.1: Obstacle and platform behaviour for accuracy tests

Figure 7.2: KF: Prediction with no noise
3.32m, the longitude's to 4.24m and the altitude's to 0.84m. Figure 7.3 shows the predicted obstacle path from noisy observations (dotted line) compared to the actual path of the obstacle (solid line). A bias can be seen in the noisy data, the predicted locations are consistently to the south of the actual locations. This is likely due to the effect of the non-linear transformation of the obstacle data from UAV centric coordinates to latitude-longitude based coordinates in the tracking module, as discussed in section 5.2.2. KFs are designed to work with Gaussian noise, and while the noise injected by the simulator is Gaussian, the non-linear conversion that has been applied means that it is no longer Gaussian when the data is passed to the predictor.

![Figure 7.3: KF: Prediction with noise](image)

The EKF based predictor, when run against the accuracy test scripts with no noise, produces a location prediction with an RMSE of 2.86m for the latitude, 2.85m for the longitude and 0.00018m for the altitude. The predicted obstacle path from the noiseless observations (dashed line) is plotted against the actual obstacle path (solid line) in Figure 7.4

When the accuracy test scripts with standard noise are run, the average RMSE is increased slightly. The RMSE for the latitude is 3.35m, for the longitude it is 4.25m and for the altitude 0.91m. Figure 7.5 shows the predicted obstacle path (dotted line)
produced from noisy data compared to the actual obstacle path (solid line). As with the KF based predictor the predicted locations show a clear bias to the south of the actual locations. Like the KF, the EKF is also designed for Gaussian noise and the obstacle coordinate transformation means that the noise is no longer Gaussian.

The KF based predictor has a much better accuracy than the EKF when run against noiseless data. However, both predictors have similar behaviour when dealing with noisy data. The difference in noiseless performance was discovered to be because of the default initial noise covariance assumed by both filters. The EKF based predictor assumes a slightly higher covariance. This perturbs the data produced by the EKF when there is no noise. This does not affect the results from the noisy data though as the noise is much larger than the assumed initial covariance.

This suggests that tuning of the KF and EKF initialization values should be looked into.
7.3.2 Convergence

The time and number of measurements required for a filter’s predicted output to converge on the monitored obstacle’s position and velocity are examined. The faster a filter’s output converges, the sooner good decisions can be made based on the filter’s output. As the OTC system is providing information to a mobile sensor platform that will probably be used for navigation and obstacle avoidance, the filter’s output must converge fast enough that there is time to avoid potential collisions.

The speed of convergence is measured by comparing the OTC system output to the known virtual obstacle positions. The output and script timestamps are then used to determine how long it takes for the absolute difference between the filter’s output and the actual value to pass below a threshold. The value for this threshold depends on how closely the receivers of the OTC system output need the output to match the actual positions as determined by the RMSE of the predictions. For example a system that can make reasonable decisions with an uncertainty of ±5 meters has a higher threshold than one that needs an uncertainty of at most ±2 meters. The threshold chosen for the tests conducted here was ±6.7 meters (GPS accuracy).

The time required for the absolute difference between the predicted and actual
locations to pass below the threshold also shows the number of observations needed. The number can be determined by counting the number of observations in the script between the first detection of the obstacle and the time when the threshold is passed.

The convergence tests are done using the accuracy test scripts. Testing the speed of convergence requires no special behaviour on the part of the obstacles or platform, so the basic motion in the accuracy test scripts is usable for the tests.

When using the implemented KF the predicted obstacle locations converge quite quickly with the actual locations. The RMSE error of the predicted location drops below 6.7m within two prediction cycles. Table 7.2 shows the RMSE of the predictions. The first prediction cycle has no data because it is used to initialize the filter data and does not produce publishable data.

Table 7.2: KF: Prediction convergence.

<table>
<thead>
<tr>
<th>Prediction cycle</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lat. (m)</td>
</tr>
<tr>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>2.33</td>
</tr>
<tr>
<td>3</td>
<td>2.46</td>
</tr>
</tbody>
</table>

The EKF predictor displays similar behaviour. It produces useful data after only two prediction cycles, and after two successive observations. Table 7.3 shows the EKF predictor's behaviour.

Table 7.3: EKF: Prediction convergence.

<table>
<thead>
<tr>
<th>Prediction cycle</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lat. (m)</td>
</tr>
<tr>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>4.40</td>
</tr>
<tr>
<td>3</td>
<td>4.31</td>
</tr>
</tbody>
</table>
CHAPTER 7. OTC TESTS  7.3. PERFORMANCE MEASURES AND TESTS

7.3.3 Time and Resources

The amount of computing resources used by the OTC system and a particular prediction and filter method are measured by memory usage, and cycles required for completion.

The cycle count is measured over critical sections to determine the length of the critical sections, which is necessary to perform schedulability analysis. Cycle counts are also measured for different numbers of tracked obstacles to determine how the filtering algorithm performance changes.

Cycle count is used as the measurement for two reasons. First, it should be consistent across machines using the same architecture. Second, the equipment available for testing was not a real-time system; therefore, while specific measures of time are not necessarily accurate, the relative performance of a filter tracking increasing numbers of obstacles can be determined.

The measurements are taken for 1, 2, 12 and 24 tracked obstacles. Currently the OTC system tracks a maximum of 24 obstacles, and it was felt that the spread of measurements would give a good indication of how the filtering algorithms scaled. The maximum of 24 is arbitrary, as there is no information on what the target system could expect under field conditions. Twenty-four was deemed to be large enough to be useful but not so large as to be difficult to work with. The maximum number of tracked obstacles can be changed easily by adjusting the OTC system configuration file and restarting the system, see section 5.5.

The RAM used by the OTC system is also measured, as is the size of the data structure used by the current predictor. This information allows the amount of required RAM to be determined should the maximum number of tracked obstacles be changed.

**RAM**

When built with the implemented KF predictor, the OTC system’s size in RAM is 231,465 bytes or 227 KB. The data structure holding the persistent filter data for all twenty-four obstacles is 3360 bytes, with each element being 140 bytes.

When built with the EKF predictor, the OTC system’s size in RAM is 232,354
bytes, approximately 227 KB. The data structure that contains the persistent filter data for all twenty-four tracked obstacles is 3360 bytes, and each element is 140 bytes.

The EKF is slightly larger than the KF which was expected as it is performing more computations.

**Prediction Times**

With the KF predictor, noiseless data produced a linear increase the prediction time directly related to the number of obstacles being tracked. The average prediction times for the noiseless data are presented in table 7.4. It shows a steady increase in the amount of time required to finish a prediction loop as seen in figure 7.6. The plot in figure 7.6 has a slope of 154 μsec per obstacle.

There is, however, an apparent issue with the performance when there are few obstacles. The plot in figure 7.6 is not linear when the number of obstacles is low, and the standard deviation for these obstacle counts is high, over 40 percent of the average value. This low obstacle count issue can be seen in all of the prediction tests and is discussed below.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Prediction Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>316.1</td>
<td>141.2</td>
</tr>
<tr>
<td>2</td>
<td>481.8</td>
<td>241.3</td>
</tr>
<tr>
<td>12</td>
<td>1954.5</td>
<td>4.1</td>
</tr>
<tr>
<td>24</td>
<td>3859.8</td>
<td>13.1</td>
</tr>
</tbody>
</table>

The same tests run with the standard noise level produce similar results. The KF predictor's performance is seen in table 7.5. The plot of the data, figure 7.7, has a slope of 140 μsec per obstacle. Again the low obstacle count issue is apparent.

Based on the observed standard deviations and examining the raw data, a difference of 10 μsec between the prediction times per obstacle for the noiseless and noisy performance data does not seem to be significant. The behaviour of the KF filter in both cases is consistent.
Number of Obstacles vs. Avg Prediction time

Figure 7.6: KF: Obstacles vs. prediction time without noise

Table 7.5: KF: Prediction time with noise

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Receive Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>371.4</td>
<td>140.3</td>
</tr>
<tr>
<td>2</td>
<td>721.1</td>
<td>168.1</td>
</tr>
<tr>
<td>12</td>
<td>1942.5</td>
<td>23.8</td>
</tr>
<tr>
<td>24</td>
<td>3691.4</td>
<td>96.1</td>
</tr>
</tbody>
</table>
CHAPTER 7. OTC TESTS 7.3. PERFORMANCE MEASURES AND TESTS

Figure 7.7: KF: Obstacles vs. prediction time with noise

The average prediction times for noiseless data compared to the number of obstacles tracked for the EKF based predictor is presented in table 7.6. Plotting this data against the number of obstacles shows that the prediction time increases linearly with the number of obstacles, figure 7.8. The slope of the line is 264 µsec per obstacle. As with the KF based predictor the low obstacle count issue can be seen.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Prediction Time (µsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>323.0</td>
<td>19.8</td>
</tr>
<tr>
<td>2</td>
<td>634.8</td>
<td>63.4</td>
</tr>
<tr>
<td>12</td>
<td>3238.9</td>
<td>2.8</td>
</tr>
<tr>
<td>24</td>
<td>6427.4</td>
<td>16.9</td>
</tr>
</tbody>
</table>

Table 7.6: EKF: Prediction times with no noise.

Running the tests again for the EKF predictor, with the standard noise level produces similar results to the noiseless tests as seen in table 7.7. The plot of the data, figure 7.9, is linear and has a slope of 262 µsec per obstacle.

As with the KF predictor, the increase in time per obstacle for the noiseless and
noisy data is small enough that the EKF predictor's behaviour is consistent across both tests.

Table 7.7: EKF: Prediction time with noise.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Prediction Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>328.5</td>
<td>21.6</td>
</tr>
<tr>
<td>2</td>
<td>620.9</td>
<td>57.3</td>
</tr>
<tr>
<td>12</td>
<td>3357.7</td>
<td>250.5</td>
</tr>
<tr>
<td>24</td>
<td>6348.4</td>
<td>76.9</td>
</tr>
</tbody>
</table>

As expected the KF predictor outperforms the EKF predictor when it comes to the average amount of time required to make predictions. The EKF is approximately 1.8 times slower than the KF per obstacle. Neither of the predictors is affected by noise in the data which is reasonable. Noise does not affect the number of computations performed.

One worrying trend in the data is the low obstacle count issue. The average
prediction times for small numbers of obstacles to have a high standard deviations. In some cases the standard deviation was more than 50.

The most likely explanation of this behaviour is preemption of the OTC system thread. The predictor runs in a critical section, however; that critical section is only protected against preemption by the other threads that make up the OTC system itself. Other processes running on the test system can preempt the OTC system, and therefore; the predictor. One likely source of this preemption is the simulator, which will be conducting file I/O on the system to read the script file.

If the number of obstacles being tracked is small fewer predictions are made overall and measurements of preempted prediction cycles are likely to make up a greater part of the total measurements. The more obstacles there are the more likely its non-preempted predictions will be measured. This would reduce the relative amount of preempted predictions as compared to the total number of measured predictions. This would mean that the effect of preemption on the average measured value and standard deviation would be larger when there are fewer obstacles and smaller when there are many obstacles.

Further work needs to be done to characterize this behaviour and determine its
CHAPTER 7. OTC TESTS  7.3. PERFORMANCE MEASURES AND TESTS

causes. This can be done in two major ways. First, by adding more tests with increasing numbers of obstacles, for example 1,2,3,4,5... obstacles instead of the current 1,2,12,24 obstacles. Additionally the profiling methods developed by George Ke in his Master's Thesis, A Path Network Model for Predicting Performance of a Loosely Coupled DRE System in the GeoSurv II UAV Project, are well suited to developing a model of the OTC system's behaviour as part of a larger computer system.

Matching Times

For the KF based predictor with the implemented matcher, the average time required to match the observations from the obstacle detector is presented in table 7.8. While the plot, figure 7.10, generally increases with the addition of more obstacles there is a spike in the amount of time required when the second obstacle is added.

Table 7.8: KF: Matching times without noise.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Match Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>81.6</td>
<td>65.3</td>
</tr>
<tr>
<td>2</td>
<td>185.9</td>
<td>118.9</td>
</tr>
<tr>
<td>12</td>
<td>228.9</td>
<td>9.8</td>
</tr>
<tr>
<td>24</td>
<td>424.4</td>
<td>63.7</td>
</tr>
</tbody>
</table>

When noise is added to the data for the KF predictor, the average time required to match the observations from the obstacle detector can be seen in table 7.9. The plot of this data in figure 7.11 shows the same spike at the addition of the second obstacle as seen with the noiseless data. The times required are quite similar, including the standard deviations.

For the EKF predictor and matcher, the average time required to match noiseless observations from the obstacle detector is presented in table 7.10. The plot of this data in figure 7.12 and has the same spike in time required when a second obstacle is added.

When noise is added to the observations for the EKF predictor the average times
Figure 7.10: KF: Obstacles vs. match time without noise

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Match Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79.0</td>
<td>63.7</td>
</tr>
<tr>
<td>2</td>
<td>156.4</td>
<td>125.8</td>
</tr>
<tr>
<td>12</td>
<td>233.6</td>
<td>16.0</td>
</tr>
<tr>
<td>24</td>
<td>461.8</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Table 7.9: KF: Match times with noise.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Receive Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>103.2</td>
<td>62.7</td>
</tr>
<tr>
<td>2</td>
<td>194.0</td>
<td>129.4</td>
</tr>
<tr>
<td>12</td>
<td>227.7</td>
<td>4.7</td>
</tr>
<tr>
<td>24</td>
<td>428.6</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 7.10: EKF: Match times with no noise.
Figure 7.11: KF: Obstacles vs. average match time

Figure 7.12: EKF: Obstacles vs. match time without noise.
to match them can be seen in in table 7.11. Once again when the second obstacle is added there is a large spike in the amount of time required to perform the matching.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>Average Receive Time (μsec.)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>104.0</td>
<td>61.6</td>
</tr>
<tr>
<td>2</td>
<td>150.2</td>
<td>126.6</td>
</tr>
<tr>
<td>12</td>
<td>231.9</td>
<td>3.3</td>
</tr>
<tr>
<td>24</td>
<td>458.1</td>
<td>27.4</td>
</tr>
</tbody>
</table>

Table 7.11: EKF: Match times with noise.

As with the predictions, noise does not affect the average matching times. There is only one matcher implemented and it performs equally well, and with similar speed for both of the predictors. It appears that as long as the predictors are producing accurate predictions they have no affect on the difficulty of the matcher’s job.

A large increase in the average matching time can be seen when a second obstacle is added. This increase is not linear with any of the other matching times. The
standard deviations for receive times with few obstacles are also much higher than
the standard deviations for receive times with many obstacles. This behaviour is also
seen with the prediction times, and should be investigated further using the same
techniques.

7.3.4 Coverage

As discussed earlier, coverage is measured with an $F_1$ score calculated from the recall
and precision values. Recall is the ratio of the number of correct correspondences to
the number of actual correspondences. Precision is the ratio of the number of correct
correspondences to the number of established correspondences.

Recall can be calculated from the number of correct predictions produced by the
OTC system and the number of observations in the simulation script file. Precision
can be calculated from the number of correct predictions made and the number of
observations received by the OTC system. From there an $F_1$ score can be calculated
and the coverage quantified, as shown in equation 3.3.

The accuracy test scripts are used to calculate the coverage of the two predictors.
The coverage test does not require specific behaviour by the obstacles being tracked,
or by the platform. Using the accuracy test scripts saves what would be a duplicated
effort of recreating simple scripts. As discussed in section 7.3.1 the accuracy test
scripts have a total of 50 obstacle observations each.

The precision of the KF based predictor is 0.71, as summarized in table 7.12.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct predictions</td>
<td>35</td>
</tr>
<tr>
<td>Received observations</td>
<td>49</td>
</tr>
<tr>
<td>Possible observations</td>
<td>50</td>
</tr>
<tr>
<td>Recall</td>
<td>0.71</td>
</tr>
<tr>
<td>Precision</td>
<td>0.70</td>
</tr>
<tr>
<td>$F_1$ Score</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Table 7.12: KF: Coverage variables.

The precision of the EKF predictor is 0.69, as summarized in table 7.13.
The $F_1$ scores of the two predictors differ by only 0.02. This gives them a very similar coverage as expected. They also have fairly high $F_1$ scores, indicating a good coverage. This means that the estimates are tracking the obstacles and that the obstacles that are present are being tracked.

### 7.3.5 False Identification

Four different false identification scenarios were touched on in section 3.5: False negative obstacle detections (missed observations), false positive obstacle detections, FIT, and FIO. As discussed in section 5.4 false positive obstacle observations from the detector are handled via the confidence value.

FIT and FIO errors are not of great interest to the OTC system as currently implemented. The GeoSurv II UAV is concerned with the presence of obstacles not their identities. The bounding box information from the detector could be useful for identifying obstacles but would likely be dependent on obstacles having consistent sizes and being different sizes from each other.

### 7.3.6 Scenarios

Three scenarios have been developed to test the OTC system’s behaviour. The scenarios involve gaps in observations, two obstacles converging and passing very close to each other, and an obstacle moving in sinusoidal motion.
CHAPTER 7. OTC TESTS  7.3. PERFORMANCE MEASURES AND TESTS

Observation Gap

By modifying the base accuracy test script to include a gap in the observations, the ability of the system to retain the track of an obstacle is tested. If the predictions are accurate, then the OTC system will be able to reacquire the obstacles. Figure 7.14 shows the platform behaviour and observations for this test.

In the gap test the obstacle moves north-west at a constant speed of 20m/s, and in the middle of script a certain number of observations which do not contain the obstacle. These empty observations are still published so that the OTC system will not report a missing obstacle detector error. The number of blank observations is chosen to create a gap of a specific size. For the tests reported here a gap of 40 meters was chosen.

Table 7.14 shows a section of the output of the KF predictor for the gap script. The drop in the confidence value shows where direct observation was lost, and the increase where observation was re-acquired. Figure 7.15 shows the OTC system output (dashed line) and the observations for the 40 meter gap script (solid line).

A section of the output from the EKF predictor is presented in table 7.15. The loss of observation can be seen in row 3 where the confidence value begins to drop because
Table 7.14: KF: OTC system output with a 40m gap.

<table>
<thead>
<tr>
<th>Lat.</th>
<th>Lon.</th>
<th>Alt.</th>
<th>Class</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.451389</td>
<td>75.701187</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451450</td>
<td>75.701279</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451641</td>
<td>75.701462</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.69</td>
</tr>
<tr>
<td>45.451736</td>
<td>75.701553</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.68</td>
</tr>
<tr>
<td>45.451771</td>
<td>75.701736</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451839</td>
<td>75.701813</td>
<td>10.00000</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Figure 7.15: KF: 40m observation gap
of the increasing time since the last observation. The re-acquisition of the obstacle can be seen in row 5 when the confidence value begins to increase again. Figure 7.16 shows the predicted path (dashed line) compared to the observations produced by the script (solid line).

Table 7.15: EKF: OTC system output with 40m gap

<table>
<thead>
<tr>
<th>Lat.</th>
<th>Lon.</th>
<th>Alt.</th>
<th>Class</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.451401</td>
<td>75.701218</td>
<td>9.999887</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451466</td>
<td>75.701309</td>
<td>10.000163</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451622</td>
<td>75.701492</td>
<td>9.999820</td>
<td>DYNAMIC</td>
<td>0.72</td>
</tr>
<tr>
<td>45.451717</td>
<td>75.701584</td>
<td>9.999784</td>
<td>DYNAMIC</td>
<td>0.69</td>
</tr>
<tr>
<td>45.451782</td>
<td>75.701759</td>
<td>10.000009</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
<tr>
<td>45.451847</td>
<td>75.701851</td>
<td>10.000208</td>
<td>DYNAMIC</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Both of the filters performed equally well. They were able to predict the location of an obstacle even after they had stopped receiving updated observations, and they were able to correctly re-acquire the obstacle once new observations were available.
Two Converging Obstacles

Two obstacles crossing each other’s paths show how the OTC system responds to situations where FIT and FIO errors are possible. The behaviour of the obstacles and platform in the converging paths scripts is illustrated in figure 7.17. One obstacle moves to the north-west and the other obstacle moves to the north-east, both moving at 20m/s.

When the KF predictor is run on the converging paths scripts the following behaviour is seen. As the two obstacles move closer together and enter each other’s matching radius the matcher confuses the two obstacles and the predicted obstacles end up follow the wrong actual obstacle, a FIO error. This is illustrated in figure 7.18. Tracked obstacle one (dotted line) changes its apparent path to match the original path of tracked obstacle two (dashed line) when the two obstacles enter each other’s matching radii.

Identical behaviour is seen with the EKF predictor, as seen in figure 7.19. The first obstacle’s (dotted line) path changes to follow that of the second obstacle (dashed line).
CHAPTER 7. OTC TESTS  7.3. PERFORMANCE MEASURES AND TESTS

Figure 7.18: KF: Converging obstacles

Figure 7.19: EKF: Converging obstacles
CHAPTER 7. OTC TESTS 7.3. PERFORMANCE MEASURES AND TESTS

This behaviour is acceptable for the GeoSurv II UAV, as the actual obstacles are still tracked and it is possible for the autonomy software to avoid them safely. However, as discussed in section 5.3.1, the matching algorithm is naive. It accepts the first match between an obstacle observation and an entry in the known obstacles list and does no further work. The matcher could be improved by checking for the best match instead of accepting the first match, and additionally by using bounding box information in the matching. This would likely reduce the occurrences of FIO errors.

Sinusoidal Motion

Testing with the platform following an obstacle that is on a sinusoidal path shows the predictors and matcher abilities to deal with non-linear motion. This is illustrated in figure 7.20.

![Figure 7.20: Sinusoidal obstacle motion](image)

The KF predictor, when provided scripted data with no noise, had RMSEs of the predictions of 21.66 meters latitudinally and 20.23 meters longitudinally. A comparison of the actual path (solid line) and the predicted path (dashed line) of the obstacle can be seen in figure 7.21. When ten scripts with standard noise included in them
were used, the average RMSE of the predictions was 22.38 meters latitudinally and 20.85 meters longitudinally.

![Figure 7.21: KF: Sinusoidal motion.](image)

The EKF predictor produced predicted obstacle locations with a RMSE of 23.37 meters latitudinally and 20.69 meters longitudinally when there is no noise in the scripts, and 22.55 meters latitudinally, 21.92 meters longitudinally when the scripts have the standard amount of noise applied. The predicted path (dashed line) and actual path (solid line) can be seen in figure 7.22.

In both cases the error from the sinusoidal motion completely swamps the error introduced by the noise. Both predictors had similar RMSEs for their location predictions and their RMSEs were also larger than the change in the actual obstacle path, whether or not noise was present in the measurements.

While both predictors were able to give an approximate location of the obstacle, its motion overwhelmed the noise in the system and neither predictor could provide a very accurate measure of the position. Their errors being approximately three times that which could be expected from commercial GPS. A tracking system requiring less uncertainty would not be able to use either of these predictors if the obstacles it expected to track had small changes to its path such as those in the sinusoid script.
Figure 7.22: EKF: Sinusoidal motion.

Given that both predictors were able to follow a single change in direction in the converging obstacles test, it seems likely that both predictors would be able to more accurately follow the sinusoid if the wavelength were longer and the peak to peak distance larger. This would give the obstacle’s path longer linear sections which would give the predictors time to stabilize after each change in direction.

7.3.7 Noise Sensitivity

Different levels of noise may affect the tracking system’s performance, therefore predictors must be tested against noise, preferably against those kinds it would encounter in the field and against varying levels of noise.

Noise sensitivity is measured by comparing the accuracy of predictors when they are run against scripts with different amounts of noise added in. The noise sensitivity tests use the same base script as the accuracy test scripts, see figure 7.1. The standard level of noise used for the tests is defined in section 7.1. Five sets of scripts were generated for the noise sensitivity tests. Each set of scripts doubles the level of noise: 0.25x standard, 0.5x standard, 1x standard, 2x standard and 4x standard.
The computational resource measurements do not affect the noise sensitivity of a prediction method, as can be seen in section 7.3.3. There is no statistical difference between the amount of computational resources used at varying levels of noise. Therefore, it is safe to disregard these measurements when computing the noise sensitivity of a prediction method.

With the KF predictor, as seen in table 7.16, the RMSE of the predicted obstacle position increases as the noise increases. The rate of increase of the RMSE for latitude is 5.87 meters per doubling of the noise level, the rate for longitude is 4.81 and for altitude it is 0.94.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Latitude (m)</th>
<th>Longitude (m)</th>
<th>Altitude (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25x</td>
<td>2.45</td>
<td>1.96</td>
<td>0.23</td>
</tr>
<tr>
<td>0.50x</td>
<td>2.43</td>
<td>3.22</td>
<td>0.43</td>
</tr>
<tr>
<td>1.00x</td>
<td>3.32</td>
<td>4.24</td>
<td>0.84</td>
</tr>
<tr>
<td>2.00x</td>
<td>6.95</td>
<td>11.23</td>
<td>1.80</td>
</tr>
<tr>
<td>4.00x</td>
<td>24.11</td>
<td>19.56</td>
<td>3.73</td>
</tr>
</tbody>
</table>

As with the KF, when the EKF is run against varying levels of noise the RMSE of the predicted obstacle position increases as the noise increases. This is shown in table 7.17. The rate of increase of the RMSE for latitude is 6 meters per doubling of the noise level, the rate for longitude is 4.96 and for altitude it is 1.03.

The EKF appears to be slightly more sensitive to noise than the KF. Additionally, neither predictor was able to produce GPS level accuracy when confronted with four times standard noise. That both predictors' accuracy worsened when the noise in the data increased seems to be a reasonable result.
Table 7.17: EKF: RMSE vs. noise levels

<table>
<thead>
<tr>
<th>Noise</th>
<th>RMSE</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latitude (m)</td>
<td>Longitude (m)</td>
<td>Altitude (m)</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>1.98</td>
<td>1.41</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>0.50</td>
<td>2.43</td>
<td>3.22</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>1.00</td>
<td>3.35</td>
<td>4.25</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>2.00</td>
<td>7.96</td>
<td>12.25</td>
<td>1.78</td>
<td></td>
</tr>
<tr>
<td>4.00</td>
<td>24.16</td>
<td>19.65</td>
<td>4.11</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 8

Conclusions

The state of the art of obstacle tracking systems was examined, and two areas for potential advancement were identified: the development of a framework for tracking method evaluation and the calculation of tracker confidence. This section summarizes the attempt to advance the state of the art concerning those areas, and presents the conclusions drawn from the development and testing of the OTC system as well as areas of future work regarding the development of the OTC system in general and the GeoSurv II UAV’s obstacle tracking system in particular.

8.1 Conclusions

A mobile sensor platform generally requires the ability to detect and avoid obstacles. Also the ability to track obstacles and predict their future locations is useful for path planning.

It is difficult to compare different tracking methods because there is no standard set of data to use or standard criteria for evaluation. This also means that an off the shelf solution to the obstacle tracking problem is difficult to obtain.

A framework testbed for tracking, and just tracking, can be used to quantify the attributes of algorithms and determine their relative advantages and disadvantages. Such a general tracking tool provides a reusable tool for the development of mobile sensor platforms, reducing the cost of such development. Additionally, a tracking
testbed allows tracking methods to be compared so that the most appropriate method for a given platform can be determined.

The OTC system as described in section 5 is a framework tracking system. It serves as a testbed for matching and prediction algorithms by providing a method to treat those algorithms as interchangeable black boxes.

To demonstrate that the OTC system works, and to show how the OTC system can be used to compare different tracking methods a set of simulation tools was created, as presented in section 6, and a set of six tests and associated performance metrics were developed in section 7.

8.2 Contributions

The development of the OTC system makes two contributions to general engineering knowledge. Firstly it demonstrates a framework obstacle tracking system that is loosely coupled to the sensors and obstacle detector. This is valuable because it reduces the cost and difficulty of developing mobile sensor platforms in the future.

Second, a method and rational for computing a self assessment of its performance, expressed as a confidence value in the data the OTC system produces is presented in section 5.4. As discussed in section 3.4 self assessment is a desirable feature of a system such as an obstacle detector or obstacle tracker. Self assessment allows other systems to evaluate the output of the self assessing system and, ideally, make better decisions than they would otherwise be capable of. However, the rational behind a system’s confidence in itself is not often presented in the literature, even when the system under discussion does perform self assessment. Thus the explanation of the OTC system’s confidence value in section 5.4 is useful both for providing a starting point for other work, and for discussion of what a tracking system’s self assessment should be.
8.3 Future Work

During the course of conducting the research presented in this theses a number of topics suitable for future research were encountered. These include, additional matcher implementations, OTC system and predictor tuning, the profiling and examination of the OTC system and its run time environment and the revision of the tests presented in section 7.

The research focused primarily on the prediction aspect of tracking as opposed to the matching of obstacles and observations. This was because the prediction was more difficult to implement correctly. However, two ideas for possible improvements to the existing matcher, or potential alternate matchers became apparent during the testing of the OTC system.

First, the implemented matcher chooses the first match detected when comparing an observation to the known obstacle list. A matcher that compared an observation to all known obstacles and chose the best match might provide an improvement in tracking. Although this would likely come at that cost of increased matcher run time. Second, the implemented matcher is a purely position based matcher. One that makes use of bounding box information, especially when two or more obstacles are in close proximity to each other, could provide more accurate tracking and reduce FIO errors. However, these improved matchers would have to be tested to determine their actual benefits or costs.

The results of several of the tests in section 7 indicate that the filters used by the predictors do not have ideal initial values or that the assumptions made for some of the default values were not optimal. Testing to determine the optimal initial and default values for the predictors would provide a better understanding of the predictor’s behaviour. Their current behaviour, if not ideal, is sufficient to demonstrate how a comparison of predictors can be carried out using the OTC system, but when development of the actual predictor method for the GeoSurv II UAV is carried out more optimal performance should be demanded of the predictors tested.

The results of the timing and performance tests in section 7.3.3 indicate that something unusual is occurring when the number of obstacles being tracked is small. A more detailed examination of how the OTC system interacts with the computer
system it is running on is called for. The profiling methods developed by George Ke in his Master’s Thesis, A Path Network Model for Predicting Performance of a Loosely Coupled DRE System in the GeoSurv II UAV Project, are well suited to this task.

An effort was made in tests to ensure that enough data was gathered to allow statistical analysis to be performed on the test results. However, the lower limits on the amount of data required for statistical analysis to be considered useful was not determined. Further experimentation to determine what volume of data is required to draw valid statistical conclusions about matcher and predictor performance would increase the OTC system’s value as a testbed.

And finally, there is the GeoSurv II UAV project itself. As test flights for the obstacle detector development are conducted, and as more of the UAV’s systems are finished and characterized, better scripts and more realistic scenarios can be developed to assist in the testing and development of the UAV’s predictor and matcher. Additionally, larger scale system tests to demonstrate the interaction between the OTC system, obstacle detector, path planner and other UAV systems will be required before the GeoSurv II UAV can begin flying missions.
References


REFERENCES


