The Pennsylvania State University

The Graduate School

Department of Electrical Engineering

TERRAIN-AIDED LOCALIZATION USING FEATURE-BASED PARTICLE FILTERING

A Thesis in

Electrical Engineering

by

Sneha Kadetotad

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2011

The thesis of Sneha Kadetotad was reviewed and approved* by the following:

Sean N. Brennan
Associate Professor of Mechanical Engineering
Thesis Advisor

Constantino Lagoa
Professor of Electrical Engineering
Thesis Advisor

Asok Ray
Distinguished Professor of Mechanical Engineering
Professor of Electrical Engineering

Kenneth Jenkins
Professor and Department Head of Electrical Engineering

*Signatures are on file in the Graduate School

ABSTRACT

The localization of vehicles on roadways without the use of a GPS has been of great interest in recent years and a number of solutions have been proposed for the same. The localization of vehicles has traditionally been divided by their solution approaches into two different categories: global localization which uses feature-vector matching, and local tracking which has been dealt with using techniques like Particle filtering or Kalman Filtering. This effort proposes a unifying approach that combines the feature-based robustness of global search with the local tracking capabilities of a Particle filter. Using feature vectors produced from pitch measurements from Interstate I-80 and US Route 220 in Pennsylvania, this work demonstrates wide area localization of a vehicle with the computational efficiency of local tracking.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are a lot of people I need to thank. Firstly I express my deep gratitude towards my advisor, Dr. Brennan, for accepting me into his research group and giving me the opportunity to pursue my thesis research in this area. His constant support and enthusiasm always kept me going. I admire his passion towards his work and love for the subject. I thank him for providing me with the required motivation whenever I felt lost and letting me make the mistakes I did to realize and discover my own way of tackling the problem at hand. I would also like to thank Dr. Lagoa for his insightful ideas and help throughout the thesis. His guidance was crucial to the completion of this work.

I am grateful to the Department of Electrical Engineering and Pennsylvania State University for providing me with the opportunity and resources to pursue this work.

I am grateful to all my teachers throughout my education for igniting the inquisitiveness present in me even today. It is this strong foundation that has helped me pursue my dreams with certainty. My deepest gratitude, love and respect towards my parents. It is only because of their upbringing and support that I confidently strode forward in my path of learning. I thank them for their never-ending patience while listening to my day-to-day problems and always showing me the right way to do things, irrespective of how hard they might be. I thank my brother for his wise words of wisdom and cheerful banter which kept me sailing in the low tides and helped me stride ahead. His view on things has always helped me make an informed decision.

I would like to thank all my lab mates for their help and support. There was never a dull moment in the lab. Pramod, thank you for your constant help and advice, which I sought literally on a daily basis. Your valuable suggestions and ideas were a big contributor to my work. I would also like to thank Kshitij for providing his insight and suggestions at times when I could not think out of the box. Sittikorn, Tejas and Jesse, you guys made every day in the lab fun. I could not have successfully completed this thesis without the support, guidance and encouragement of you all.

A special mention to my roommates and all my friends. I thank all of you for always being by my side and keeping my life balanced with all the fun and frolic. Thank you for letting me know you are always there for me and that I can depend on you all.

To my parents.

You taught me to believe in myself.

CHAPTER 1
INTRODUCTION

This chapter briefly introduces the concept of vehicle localization and the techniques currently used for detection and tracking. A more detailed study is presented in the following chapter.

1.1 Motivation

According to the U.S. Department of Health and Human Services [1], motor vehicle accidents are the leading cause of death in the United States when causes of death by disease are not included. In 2007, automobile crashes claimed 41,059 lives, and 2,491,000 people were injured in 6,024,000 police-reported motor vehicle traffic crashes. Hence the development of on-board automotive driver assistance systems that alert the driver about the driving environment as well as possible collisions with other vehicles has become of great importance. The accurate localization of road vehicles can provide knowledge of a vehicle's position at any given point in time relative to fixed objects (eg. guard rails) in the surroundings, providing hope for the development of effective collision avoidance systems.

Accurate position information is also essential for implementing *autonomous vehicle control* [2] and *driver assist systems* [3] for safety of the passengers. In these applications, position information can be used for object detection, curve warning, and even speed zone information. Accurate localization of a vehicle can also be used for parameter estimation, including road friction, tire slip and odometry bias error.

There are research efforts currently underway to develop Intelligent Vehicle Systems that are capable of self-navigation and have collision-warning systems. Sun et al [4] describe the importance of efficient vehicle detection and tracking methods and present a survey of some of the current methods being used in developing intelligent vehicles. They also elaborate on the use of radar systems and LIDAR systems as well as image processing for vehicle detection and tracking.

Current Vehicle Localization Methods

The current method to localize a vehicle is through the use of a Global Positioning System (GPS) and in the present study the position of a vehicle using a GPS system has been used as the ground truth while comparing position estimates to calculate prediction error. A schematic illustrating the GPS positioning process is shown in Figure 1. Knowledge of the mechanism of functioning of this system is essential to understand its drawbacks and the need of new techniques. Hence a brief account of the functioning of a GPS system is presented.

This system makes use of satellites orbiting the Earth along with a GPS receiver at the ground station. The GPS receiver calculates its distance from each of at least four satellites, whose positions relative to Earth are known accurately, and hence calculates its own position. The functioning of the GPS system is as follows.

The GPS receiver needs the following information to perform its calculations:

- The location of at least three satellites around it.
- The distance between it and each of those satellites.

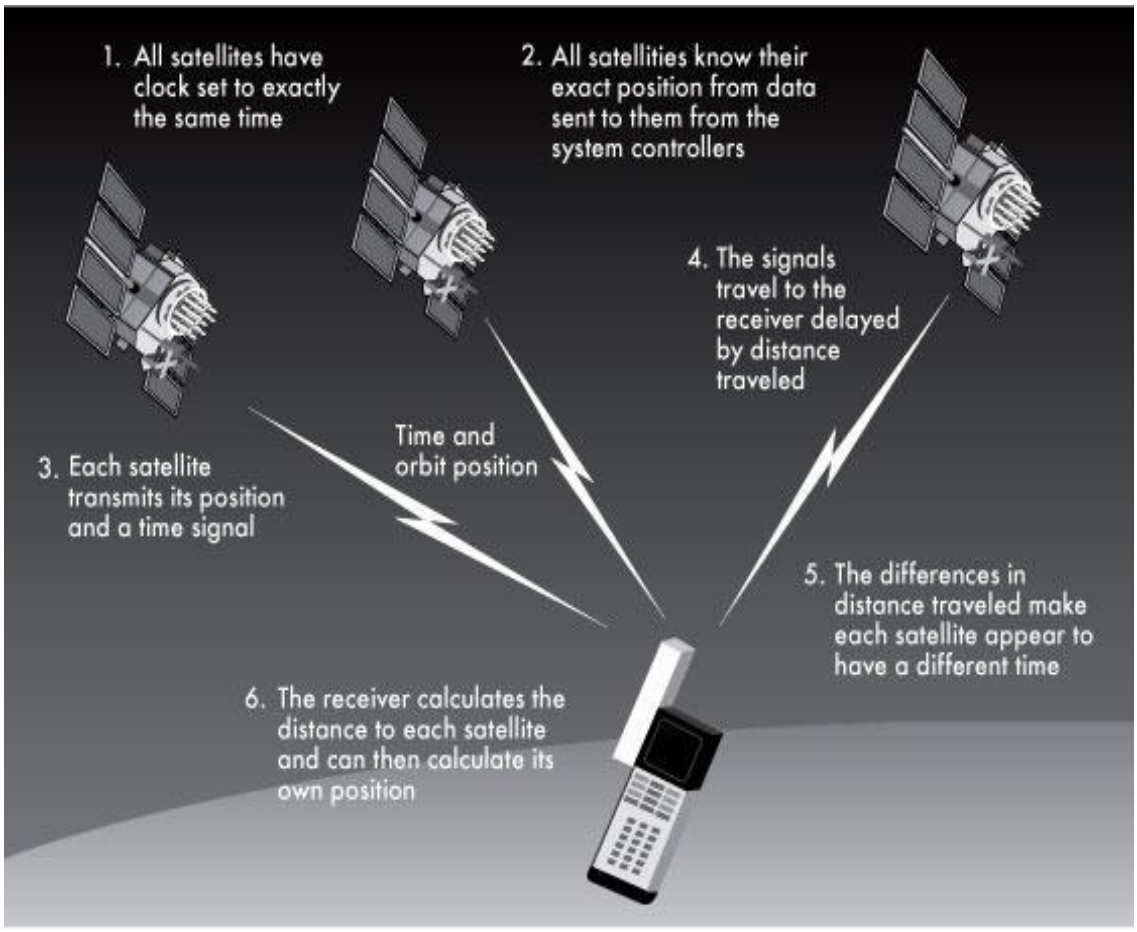

Figure 1: Functioning of a Global Positioning System [5].

At a particular point in time, each of the satellites sends a signal to the ground receiver. The ground receiver will begin running the same signal code at exactly the same time as the satellites. So when it receives a signal from one satellite, it can compare the two codes and know the time taken for the signal to travel the distance between that particular satellite and itself. Knowing the speed of light, it calculates the distance between the satellite and itself. Hence it can form a sphere of a particular radius within which it expects the satellite to be located, and with information from four

satellites it gets four spheres which intersect at one point. Knowledge of the exact location of each satellite at any given point in time is also needed. This is already known due to the predictable nature of the satellite orbits. Their positions are constantly monitored by the Department of Defense.

The GPS system described has a number of disadvantages that make it unreliable for accurate vehicle localization and have lead to research into alternate methods of localization which do not depend on satellite systems.

The GPS system that is currently used suffers from the following disadvantages:

(1) **Slow Update Rate** (around 10-20 Hz): Most of the GPS systems have a very slow update rate and hence give the driver wrong information about his location quite often [6]. In certain situations this can be detrimental to the driver's safety.

(2) **Poor Signal Reception**: GPS devices are extremely fragile and need a constant connection with the ground station and satellites to give correct information. In locations with tall buildings or tunnels, due to sparse coverage, reception can be poor and the system may not function as desired.

(3) **High Cost**: The development and functioning of the entire system can be quite expensive in terms of infrastructure and working.

(4) **Vulnerability to signal jamming:** The low power signals provided by a GPS system are highly susceptible to interference [7]. Hence service over a large area can be easily jammed. This issue is of importance even to the U.S. military as it leads to security threats during war situations.

(5) **Overall fragility of satellite constellation**

There is therefore a need for developing systems that can localize a vehicle without the use of satellites. Methods have been developed to assist GPS systems during short periods of GPS-outages, like when the vehicle goes through a tunnel. Najjar et al [8] proposed the use of wheel

odometry along with GPS. Here the distance traveled by the vehicle once the GPS stops functioning is measured, and using the last recorded position given by the GPS, the current position estimate is calculated. There are numerous issues with this method: first, it requires initialization by a GPS. Second, there can be wheel-slip errors that are not accounted for. And lastly, the sensors used for measurement can have an inherent bias error which could cause the estimated value to drift further away from the true position with time.

There are other methods to localize a vehicle, including the use of vision sensors [9] or LIDAR [10] (Light Detection and Ranging). The issues with these techniques are their extreme vulnerability to weather conditions and their tendency to fail in dusty or rainy conditions. Many vision systems cannot be used under poor lighting conditions (night-time). Overall these systems also tend to be computationally and financially expensive.

Another method that is commonly suggested is the use of beacon systems [11], where the position of the vehicle can be triangulated. The main disadvantage of this technique is that it is not accurate enough when it is implemented in real time, and is difficult to deploy without an existing beacon infrastructure.

Related to the present study, A. Dean and R. Martini [12] proposed the use of terrain related information (roll and pitch measurements) to localize a vehicle. Similar to work done in [13] where an aircraft's elevation profile is matched to a digital elevation map, the terrain map, consisting of roll and pitch data, is used for vehicle localization. Here it is assumed that the roadway has been previously mapped and the terrain information is available on-board the vehicle. A. Dean showed the efficient tracking of a vehicle using terrain information with both Kalman filtering as well as Particle filtering [14]. Local tracking refers to estimating the current position of the vehicle

using the last position estimate and the information obtained from various sensors present on-board the vehicle. An inherent assumption in this methodology is that the initial position of the vehicle is known. Methods used to track a vehicle are generally based on estimation algorithms like A. Dean's work on Particle filtering [15] and Kalman filtering [16].

Localization of a vehicle can be broadly divided into two categories based on their solution approach: Global localization and Local tracking. Global localization refers to finding the location of the vehicle when it can be present anywhere on the map and its initial position is not known. The method used to do this is through the use of feature matching [17-21], where unique features are formed from available map data and then a search is performed to locate the position. The problem of Global localization was approached by P. Vemulapalli et al [22] using wavelet based features generated from terrain data. The proposed features utilized maxima-minima points in terrain data to locate the position of a vehicle.

Given the previous methods that utilized Kalman or Particle filtering to solve the local tracking problem and the feature-based approach to solve the Global localization problem, this thesis proposes an efficient combination of the two methods: feature matching and Particle filtering, to localize and track the vehicle.

1.2 Problem Statement

The primary objective of this study is to use feature-matching techniques incorporated into estimation algorithms to efficiently localize a vehicle on the roadway. The estimation technique used will be the Particle filtering algorithm and the features utilized in this project are based on wavelet modulus maxima. Prior work in this area deals with Global localization and local tracking as separate issues, solved using different techniques. This effort proposes the use of feature-based

Particle filtering techniques to solve both the issues, that is, be able to localize and track a moving vehicle without any knowledge of its initial position.

Work done by P. Vemulapalli in the same research group focuses on detection of features in live streaming-in data and he is currently working on the optimal filter for feature detection. His work includes the detection of features based on wavelet modulus maxima. Work done in this thesis implements a particle filter re-sampled using these features to localize and track a vehicle.

1.3 Outline of Remaining Chapters

The remaining chapters are organized as follows:

Chapter 2 presents a more detailed literature review of the existing techniques for vehicle localization that are either GPS-free or can be used only to aid the functioning of a GPS during short periods of outages. These techniques typically use vision, LIDAR or terrain data to localize the vehicle. **Chapter 3** provides a broad overview of Global localization and local tracking and the main methods used to solve the problems, Particle filtering and Feature-based techniques respectively. **Chapter 4** describes the algorithm proposed in this thesis which utilizes features to localize and track the vehicle via the Particle filtering algorithm. **Chapter 5** summarizes the results of the thesis and details the advantages of the proposed method over previously used techniques. This thesis concludes with **Chapter 6** which proposes different directions for future work and other possible applications of the algorithm.

CHAPTER 2
LITERATURE REVIEW

As mentioned in the introduction, the concept of localization of an object and its tracking has been of tremendous importance in the recent past. A large part of ongoing research in the robotics domain deals with tracking and guiding the movement of a robot.

Vehicle localization has been of great importance and inspiration to many separate research efforts. The current method of vehicle localization, a Global Positioning System (GPS), has some very glaring issues with it that have prompted researchers to explore other techniques to either augment or replace GPS in outage situations.

2.1 Map-based Approach

Alternate localization methods typically utilize a map-based approach, for example Li's work in [23], in which the roadways of interest are initially mapped by collecting certain sensor data and processing this data to extract different parameters of interest. The map created in this process has a record of the parameter of interest and the locations corresponding to where these parameters were found. To perform localization, the vehicle must be outfitted with the sensor and be supplied with the map. The computer onboard the vehicle then computes the same parameter of interest (in real time) by using the sensor data streaming-in and uses the map to estimate its current position. Different researchers have used different types of sensors and different types of parameters to solve the localization problem, some of which are explained in greater detail in this chapter.

The problem of localization of a vehicle can be broadly classified into two areas based on the solution approach: global localization and local tracking. The latter is by far the most-studied problem. Here one assumes that a robot knows its initial position and "only" has to correct small errors as it moves. The sensors used to solve this problem in the map-based approach include LIDAR, Vision sensors and INS sensors. Local tracking algorithms involve a localized search, these algorithms typically utilize tracking methods such as Kalman filtering, unscented Kalman filtering [24] or Particle filtering [25].

The global localization problem involves a robot whose initial position is unknown; hence, it has to solve a much more difficult localization problem, that of estimating its position from scratch. Typically the Global localization methods utilize feature matching techniques that perform matches by using data structures such as KD-trees [26] or vocabulary trees [27] to quickly search through the map.

A very active research area at present is to find techniques to perform simultaneous mapping and localization (SLAM), that is, acquiring a map of an unknown environment with a moving robot while simultaneously localizing the robot relative to this map. For example, Thrun et al [28-30] used hierarchical techniques to reduce the size of the maps and they also proposed the use of sparse extended information filters for detection and mapping, which are significantly faster than EKF techniques. While these methods are interesting, they are much more complicated than the proposed approach due to the need to both build and use a map simultaneously.

2.2 Lidar & Vision Sensors

In a demonstration that LIDAR data alone can localize a vehicle, Bosse and Zlot from the CSIRO ICT Centre in Australia [31] used a LIDAR sensor and extracted different types of statistical

parameters from the LIDAR data to create a map and then performed Global localization. They proposed a methodology to compare the keypoints (statistical parameters) obtained off the road with those in the map using a k-nearest neighbor search. Similarly, Schindler, Brown and Szeliski [32] utilized vision sensors and image processing to extract SIFT features to globally localize themselves in an urban environment. They made use of Vocabulary trees to store features and examined methods to optimize the search results using features that are most informative about a particular location. A paper by Fontanelli, Ricciato and Soatto deals with the local tracking problem using only LIDAR measurements in combination with an Extended Kalman Filter [33]. The sensor is rigidly fixed on a generic moving platform, which is assumed to move in planar surface. They use the RANSAC algorithm in combination with a Huber kernel in order to cope with typical distortions in LIDAR measurements. The robust registration is successively used in combination with an Extended Kalman Filter to track the trajectory of the LIDAR over time, and hence solve the localization problem.

Even though successful localization has been achieved using LIDAR and vision sensors, they have a number of drawbacks. Both these sensors fail under poor weather conditions like rain and snow. They can also get easily blocked by dust, dirt or snow. Vision sensors cannot be used under poor lighting conditions, like at night-time. These sensors are also relatively expensive. Hence, the use of INS sensors to measure road grade and use of this information for localization is a better approach as it is not affected by weather conditions or lighting.

2.3 Terrain-aided approach

A novel idea was proposed by R. Martini in [34] where he showed that terrain-related information (roll and pitch values) correlates to specific locations on a roadway. Consequently the

idea of using INS sensors to collect roll and pitch values for a roadway and create a map which can then be used for localization was conceived. The advantages of using terrain data is evident because unlike LIDAR and Vision sensors, the inertial sensors are not affected by external conditions such as rain, dust, fog, visibility etc. A. Dean et al implemented local tracking algorithms like Particle filtering and Kalman filtering using terrain data to efficiently track the movement of a vehicle in [12],[15] and [16]. This work is explained in greater detail in the next section which elaborates on the Particle filtering algorithm used by them for the tracking process. This algorithm was optimized in the present work through the incorporation of features. Terrain-aided applications also include missile-guidance systems [35] and underwater robotics [36].

Vemulapalli et al worked on the Global localization problem using terrain related data. This work proposed the use of 'multi-scale features' to perform localization [22]. A database of features detected on the roadway in the mapping phase is stored in the form of a KD tree and then a search is performed to obtain a match for the feature that is detected.

Considerable work has been done in the genre of detection of features and their use in localization. Work by Ledwich and Williams [37] explains the use of SIFT features that are invariant to rotation, translation and scale variation, for matching images. Murphy et al [38] showed that by combining local and global features, we get significantly improved detection rates. Here they obtained numerous feature vectors for an image by convolving each image with a bank of filters and then extracting image fragments from one of the filtered outputs at random. Thus a lot of research is underway to improve on the feature-matching process.

The previous work in inertial data based localization, whether that be using pitch or roll information, has utilized the raw sensor data as an identifier (parameter of interest) for a location. In contrast, the present work draws upon the feature-based matching approaches proposed by

Vemulapalli et al [22] that have been built for global localization and utilizes these features, extracted from raw sensor data, as the parameters of interest. The technique of performing feature-based tracking has advantages in terms of both computation and memory. This is critical because the localization algorithms will have to perform in a real-time environment and also have reasonable memory requirements to be practicable. This effort works towards proposing a novel technique to perform Global localization and local tracking simultaneously using a Feature-based Particle filter.

CHAPTER 3
GLOBAL LOCALIZATION & LOCAL TRACKING

This chapter gives a detailed explanation of the method proposed by Vemulapalli et al [22] and A. Dean et al [25] to solve the Global localization and local tracking problems respectively. An in-depth knowledge of these techniques helps understand the functioning of the Feature-based Particle filter proposed in this thesis.

3.1 Global Localization

Global localization refers to the problem of determining the location of an object without any knowledge of its initial position. This problem is much more difficult to solve than local tracking. One approach often used to tackle this problem utilizes the concept of 'feature vectors'. Feature vectors can be defined as unique information extracted from a signal that can be used to identify or locate the signal in a vast set of similar signals.

As mentioned in [17] [18] [19], the audio track retrieval problem is a global feature matching problem that has been dealt with by the introduction of numerous 'features' to solve the map-matching problem. Vemulapalli et al deal with solving a similar map-matching problem with road pitch data in [22]. Work done by them generates feature vectors from terrain data, roll and/or pitch values, and uses it for global localization.

The process of generating 'feature vectors' from road pitch data and using them to localize a vehicle can be divided into two broad phases. The first phase, the preprocessing phase, consists of the following steps. First, collected data is passed through a transform, often a wavelet transform, to separate it into information in different frequency bands. It has been observed that

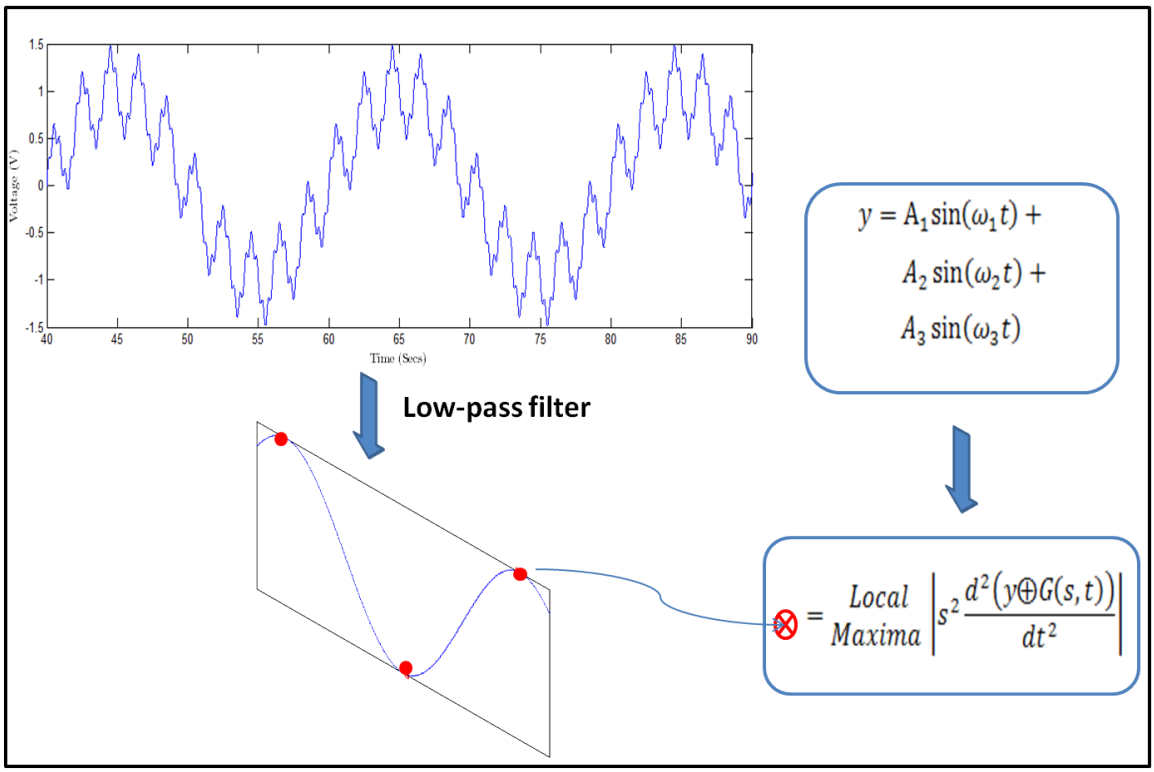

Figure 2: Procedure for feature generation.

terrain data has significant noise in high frequency regions so this process is a means of isolating signal from noise, example after separation the data in the high frequency bands is ignored. In this work, we chose the lowest frequency band signal as our signal of interest. The wavelet transform was performed using the "Gaussian wavelet". For this work a low-pass Gaussian filter was used with a cut-off frequency of .0074 cycles per meter, e.g. 1 cycle every 136 meters. This long spatial period was chosen because iterative analysis of data showed that the long-period frequencies on a roadway gave the most repeatable features above this spatial cut-off frequency.

The output signals obtained from the wavelet transform contain peaks corresponding to the high curvature points of the signal of interest. These 'key points' which are detected using second derivative transform refer to the maxima-minima points in the data. There are many ways to represent signals as features. One of the simplest is to use the maxima-minima points in a

specific frequency band as the feature points. These are calculated from the wavelet transform at the chosen scale. At a point of local maxima, say at scale *s0* and point *u0*

$$\frac{\partial \mathrm{Wf(u,s)}}{\partial \mathrm{u}} = 0|_{\mathrm{u=u0,s=s0}} \tag{1}$$

The key points are found from the finite difference implementation of Equation (1).

Each feature is represented by the value of the extrema points and the relative distance between the extrema point and its preceeding extrema, as shown in Figure 3. This use of relative distance instead of absolute distance makes the feature invariant to bias and scaling, an important property because both errors are common in field data collection. The feature values are comprised of five pitch values of the five maxima, or key points, and the four relative distances

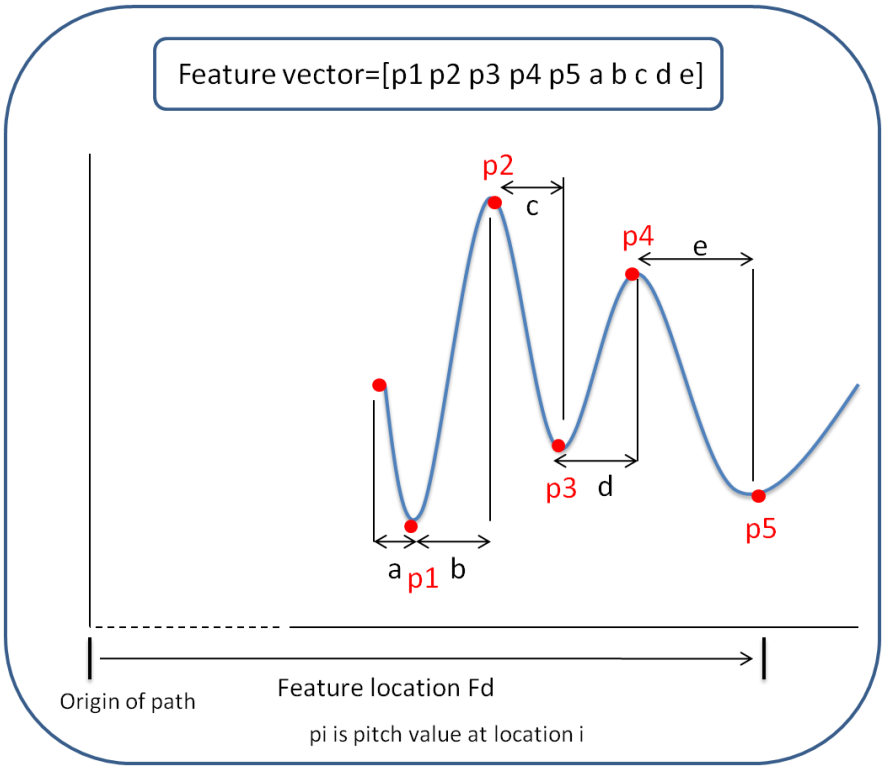

Figure 3: Feature vector measures

between the key points. Two sets of information are stored in the database corresponding to each feature: first, the above mentioned feature vector including pitch values and relative distances, and second, the location of this feature relative to the path origin and measured to the end of the feature. The latter is used as a distance measure to record the location of each feature detected.

The number of maxima-minima points chosen to form one feature vector depends on the degree of uniqueness desired. A very short feature would be less unique and therefore give little advantage over raw pitch values but its processing would be relatively fast. In contrast, a very long feature would be very unique but would require long time periods to detect vehicle position on flat roadways. This would cause a problem in quickly localizing a vehicle on certain sets of roads. The feature length of five points was chosen through tests on different road type datasets as a good tradeoff between uniqueness and convergence rates.

The previously collected pitch data is used to generate features via the process described, and these features are stored in the form of a sorted list database. This database is sorted in the feature space.

In the online phase, as the vehicle travels, pitch data is collected. The moment a feature is detected, a query feature is sent to the feature-based database. The closest match is found and from that match a position estimate is calculated.

The global search problem can be dealt with using the technique of querying a database of features. But to efficiently track an object after localizing it globally, we would require an estimation procedure which could use odometry to not only progress the position estimate forward but also to estimate the probability density function of new features. Estimation techniques

generally used to achieve both goals are Particle filters or Kalman filters. The next section describes the use of Particle filtering to track a vehicle.

3.2 Particle filtering for local tracking

The Kalman filter is an estimation algorithm for efficiently tracking a linear dynamic system. It combines a Minimum Mean Squared Error (MMSE) estimator with a model of how a Gaussian random variable propagates through a linear system. It is comprised of a predictor step where the state estimate is predicted based on previous measurements and the state model, and a corrector step wherein the estimate is corrected based on a current measurement. In this filter the mean and the covariance of the state propagate with time. The drawback of the Kalman Filter is that it is the optimal estimator only when the state model is linear and when the noise is Gaussian. It also needs to be initialized with a Gaussian probability density function, which is not a valid assumption for this work.

Alternatively, Particle filtering is an estimation technique best suited for non-linear systems. This estimator works well even with non-Gaussian distributions and does not require linearization of the state model. In this approach a set of particles represent the probability density function (pdf) of the state estimate. The particles are placed on the map initially and then weighted every time step based on the current measurement and the particle's position. The weighted particles are then re-sampled to eliminate particles with a low weight and multiply those with a high weight. Similar to a Kalman Filter, the particles are moved forward based on the state motion model. The high computational power required in Particle filtering was one of the major drawbacks associated with this technique, but the advances made in computing power in recent years allow this technique to be a feasible option even for real-time implementations.

The present work builds upon the work by A. Dean et al [12] which presented the use of a Particle filter to locally track a vehicle. In this prior study, roll and pitch information was found to closely correspond to vehicle position. The algorithm used these correlations to find the position of a vehicle in a two-stage approach: the preprocessing phase and the online phase. In the preprocessing phase, pitch and/or roll data values were collected while driving on the roadway, and then stored on board the vehicle.

The main steps involved in the online phase were as follows. Initially the map was populated by particles that were randomly placed. While driving down the roadway, pitch values were collected. The particles moved forward on the map through a propagation step using the Equation (2).

$$\mathrm{X_p^k} = \mathrm{X_p^{k-1}} + \mathrm{dX} + w \tag{2}$$

Here X_p^k is the position of the pth particle at the kth time step, *dX* is the distance the vehicle travels between iterations as inferred from odometry, and *w* is Gaussian white noise of variance Q equal to the variance of the odometry measurement.

The added Gaussian noise accounts for noise in odometry measurements and also maintains a degree of randomness in the location of the particles to prevent them from converging to any location too soon. After the update step, the particles were weighted based on the degree of match between the pitch value just collected off the roadway and the pitch value corresponding to each particle from the map database. The weighting mechanism generally chosen is a Gaussian weighting mechanism. The Gaussian is chosen to be wide enough to ensure that if a wrong estimate is given a higher weight, the correct estimate is still within the curve and is given a weight high enough to prevent its elimination during the re-sampling step. But the Gaussian cannot be so

wide as to encompass too many adjacent particles which could cause divergence in the Particle filter population. The Gaussian also affects the rate of convergence of the algorithm and should have a value that ensures the algorithm does not converge too soon by giving very high weights to the wrong particles initially. As a result of this, the variance of the Gaussian needs to be chosen very carefully. The weighting scheme used was of the form

$$q_i = \mu^{-1} \cdot \exp\left(-0.5 \cdot R_\theta{}^{-1} \cdot (\theta_a - \theta_{p,i})^2\right) \tag{3}$$

where ϑ_a is the pitch measurement, $\vartheta_{p,i}$ is the i^{th} particle's pitch corresponding to its position along the terrain map, and μ is a normalizing factor equal to the sum of the particle weights, R_ϑ is the variance of the attitude measurement.

After weighting each particle, a re-sampling step was conducted to eliminate particles with a low weight and multiply the ones with a high weight. The re-sampling step was basically a ranking scheme to give more importance to the particles which have a higher likelihood of being in the correct position. The details of the re-sampling step can be found in [15]. After re-sampling the particles, the position estimate was chosen to be a mean of the positions of all the particles. This process was performed repeatedly to localize and then track the vehicle. Figure 4 illustrates the details of the process.

The disadvantage of this previous work was that it used all the pitch values collected and therefore required a large database to store this information. It was found that much of the pitch information stored was redundant and did not help in improving the accuracy of localization. A considerable amount of computational effort was also required to accurately localize the vehicle, quantified later in terms of Floating-Point Operation Counts (FLOPS). To overcome these disadvantages and optimize the existing technique the current work proposes a Feature-based

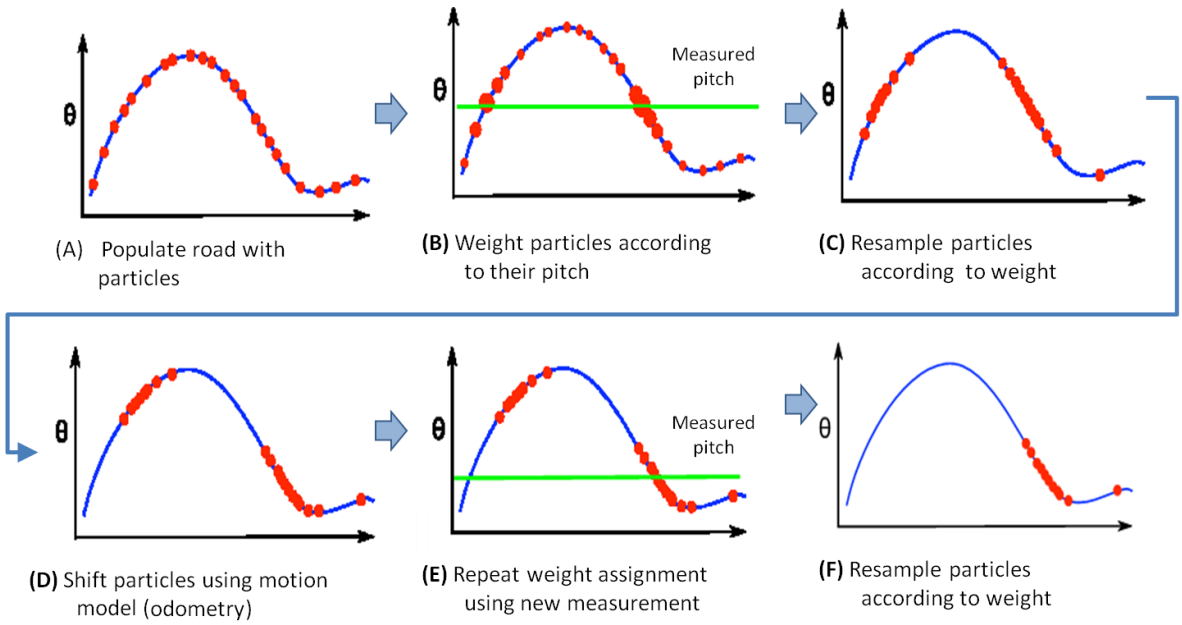

Figure 4: Particle filtering technique from [14]

Particle filter that makes use of the concept of features from Global localization to update the Particle filter. This Feature-based Particle filter is elaborated on in the following chapter.

CHAPTER 4
FEATURE-BASED PARTICLE FILTER

The Feature-based Particle filter is a novel idea that is designed to localize and track a vehicle using an efficient combination of feature generation and Particle filtering. The Particle filter can make use of the 'uniqueness' of a feature vector and operate the re-sampling step only when a feature is detected. This makes the algorithm more efficient and accurate by relying on the uniqueness of features rather than on every pitch value collected. The goal of this algorithm is to use the power of unique features to update a Particle filter. The algorithm can again be divided into two main parts: the preprocessing phase and the online phase. Figure 5 gives an explanation about the algorithms implemented in each of these phases.

Preprocessing phase:

In this phase the vehicle is made to travel on roadways and collect pitch values which are then used to generate feature vectors as described in Section 3.1. Figure 6 gives a diagrammatic explanation of this phase. The assumption made is that the roadway has previously been mapped to obtain pitch values. This is a valid assumption due to the large number of current research projects which are mapping various highways in the United States. A database is created in the feature space, which comprises of features formed by five key points, or maxima points. For each feature vector formed, the data stored in association with it is:

(1) The set of five pitch values

(2) Four relative distances of each of these pitch values from its neighbor.

(3) The location of the feature vector within the map, referenced to the last pitch value in that set.

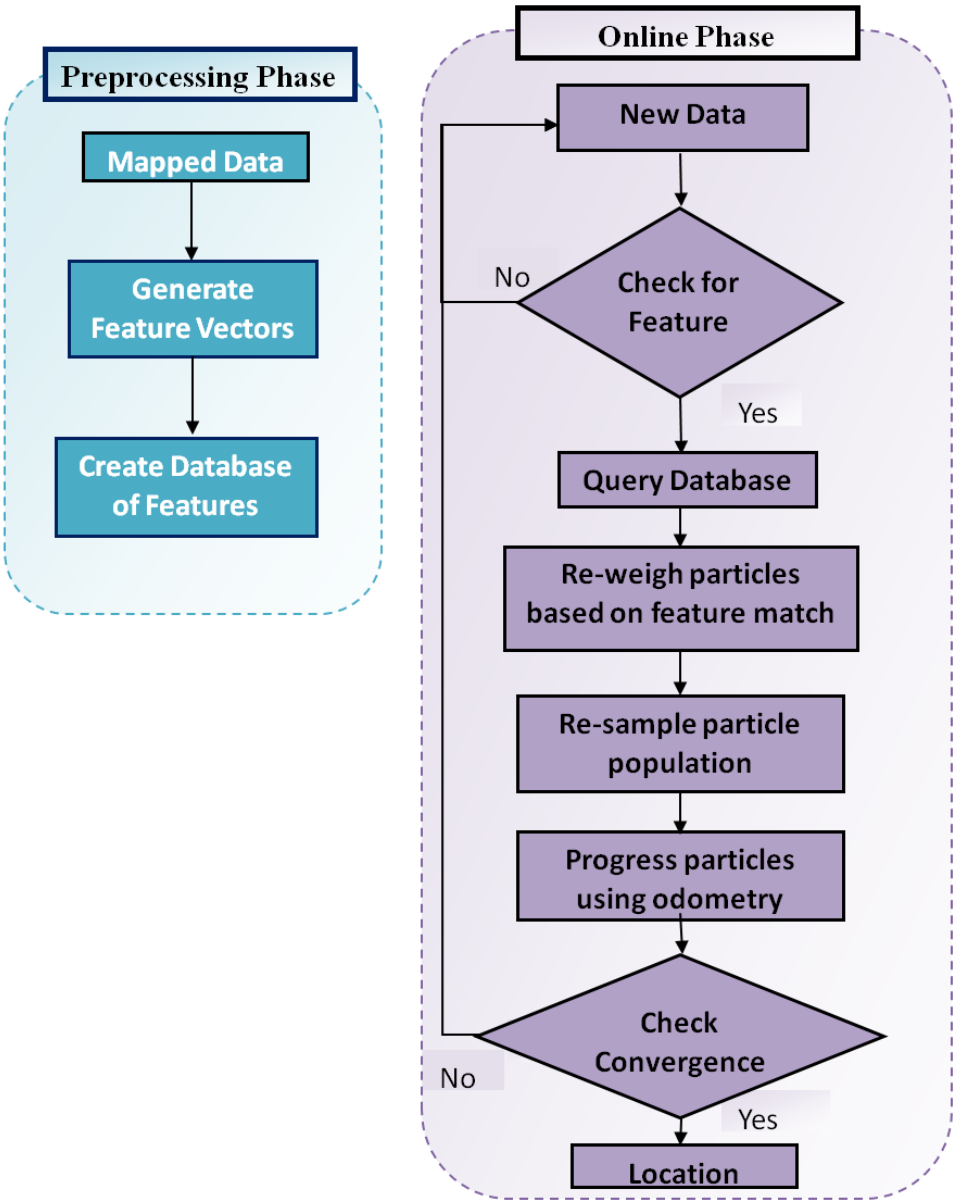

Figure 5: Feature-based Particle filter

Each feature stores information that correlates the pitch domain to the distance domain. This preprocessing phase is performed offline and the data is then stored on-board the vehicle. The

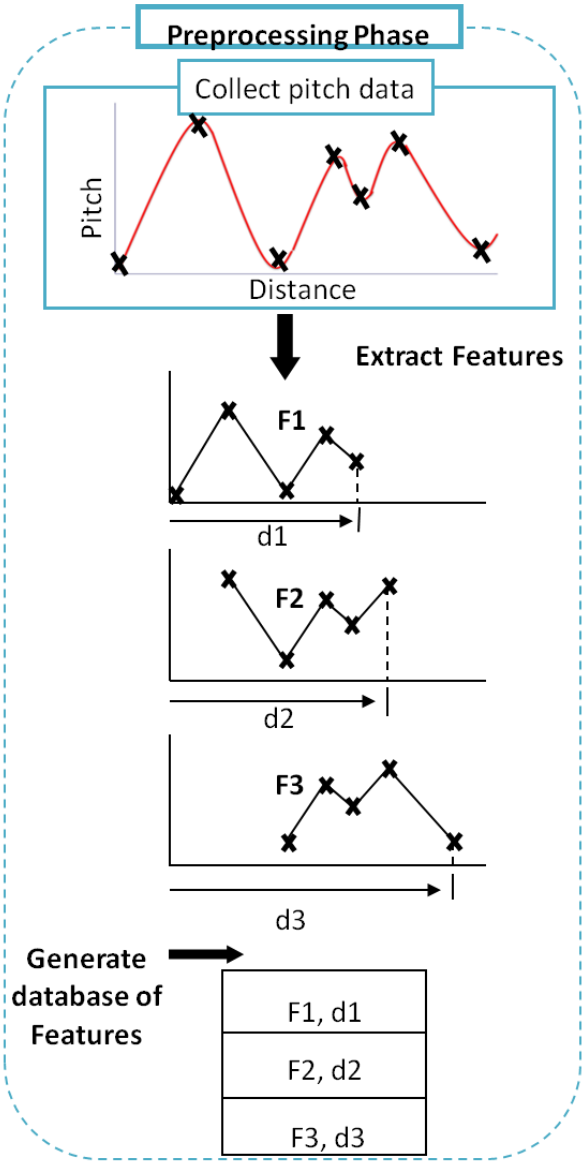

Figure 6: Preprocessing Phase

feature database requires much less memory than the database of raw pitch sequence [16] where all the pitch values are stored for each roadway. To illustrate, a comparison of memory requirements was performed examining raw pitch versus feature databases for a 11 km portion of roadway from Interstate I-80 in Pennsylvania. The classical Particle filter needed 4. 10MB when this 11 km long dataset was used, while the feature-based approach needed only 55.2 KB. This is a reduction by a factor of 75. The comparisons are elaborated on in Chapter 5.

Online phase:

Figure 7 below shows the steps of the online phase. In the online phase, the localization process is initialized in step 1 by populating the map with a set of equally weighted particles randomly placed on the map, like a typical Particle filter. As the vehicle drives down the roadway in step 2, pitch data is collected and the wavelet transform and maxima-minima detection steps are performed as described earlier. As the vehicle keeps moving forward, the particles are also propagated in the map using equation (2) as in the classical Particle filter, by a distance that is determined by odometry measurements. This is shown in step 3.

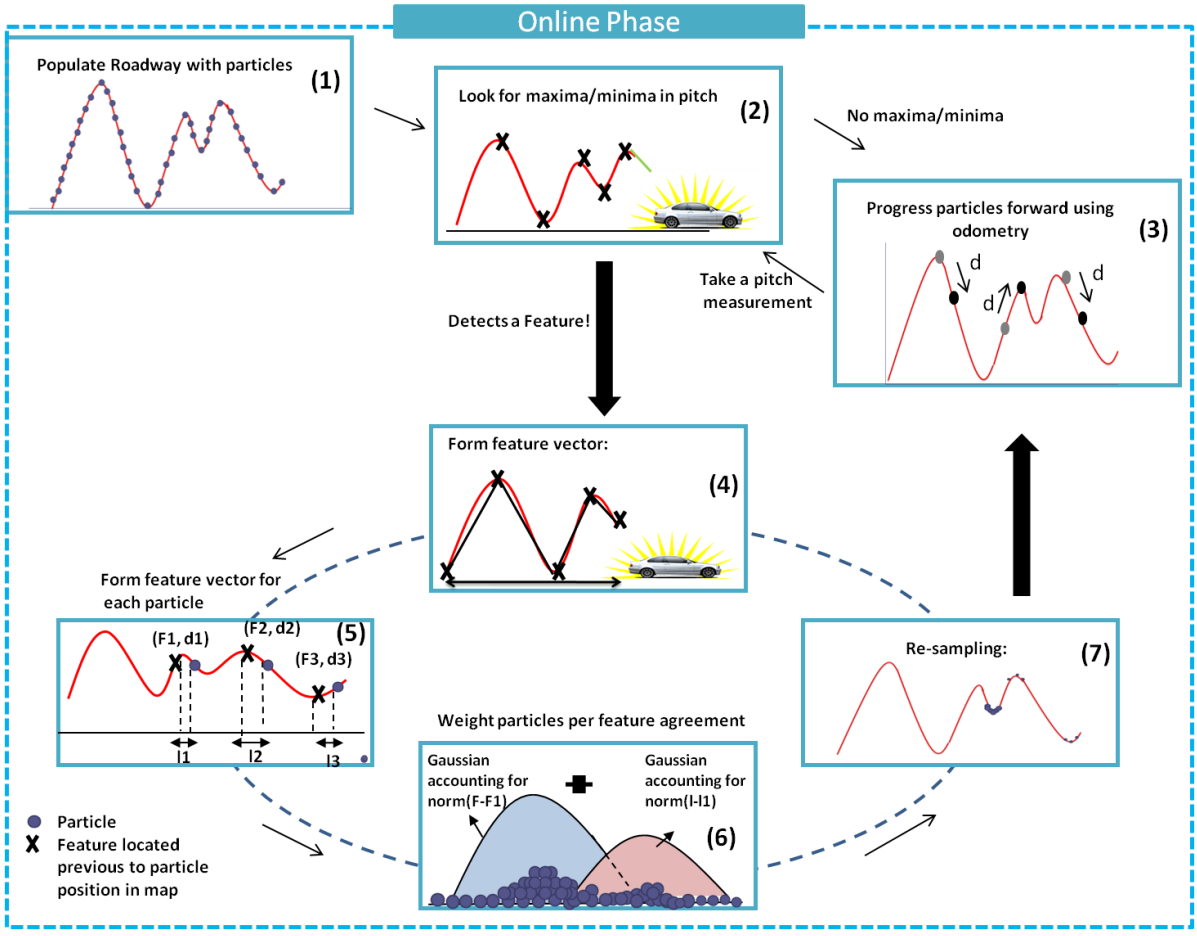

Figure 7: Online Phase

The main change introduced here is that the Particle filter correction step is not conducted unless a feature is detected; only then does it perform the re-weighting and re-sampling steps.

For each particle located on the map there is an 'associated' feature from the feature database. The 'associated' feature for each particle is defined to be the feature most recently encountered by the particle from the map database. The moment a feature vector is detected in step 4 each particle located on the map is re-weighted based on the degree of match between the measured feature and the feature 'associated' with the particle as shown in step 5. The weight given to the particle is based on two weights: a feature-matching weight and a distance-matching weight, described next.

To form the feature-matching weight for each particle, the measured feature is compared to the 'associated' feature for each particle. This is done because, if the particle is at the correct location, then the closest feature located prior to its current position should be a strong match with the measured feature. The weighting scheme used in this work is a Gaussian weighting scheme. The details about the choice of this scheme is explained in Section 5.4. The weight associated with the feature-match is obtained from a Gaussian weighting function of the form used in the classical Particle filter

$$q_i = \mu^{-1} \cdot \exp\left(-0.5 \cdot {R_\theta}^{-1} \cdot (\theta_a - \theta_{p,i})^2\right) \tag{4}$$

with the difference that instead of $(\theta_a - \theta_{p,i})$ the term used is diff(i)

$$qf_i = \mu^{-1} \cdot \exp\left(-0.5 \cdot K \cdot {R_F}^{-1} \cdot (\mathrm{diff}(i))^2\right) \tag{5}$$

where

$$\mathrm{diff}(i) = \|F(i) - F\|_2 \tag{6}$$

F(i) is the feature vector associated with the i^{th} particle, *F* is the measured feature vector, qf_i is the weight of the i^{th} particle based on feature matching and *K* is a tuning factor. In place of R_θ in

equation (4), R_F, the measurement noise variance in feature, is used. The value of measurement noise variance in pitch is known from work by A. Dean in [14]. If we assume that the pitch values are independent of each other, which is an assumption made in this case, the covariance matrix for the feature error is

$$R_F = \begin{Vmatrix} R_\theta & 0 & 0 & 0 & 0 \\ 0 & R_\theta & 0 & 0 & 0 \\ 0 & 0 & R_\theta & 0 & 0 \\ 0 & 0 & 0 & R_\theta & 0 \\ 0 & 0 & 0 & 0 & R_\theta \end{Vmatrix} \tag{7}$$

Hence the value of variance in feature error measurement is again R_θ.

Giving a weight based on the degree of match, in this case the norm of the difference between the pitch values of the two features, makes sense. If there was no feature located prior to a particle's position in the map, then that particle is unlikely to be at the correct position and so is given no weight. It then gets eliminated in the re-sampling step.

The weight based on distance match is used because, as a vehicle is traveling on a roadway, it can definitively detect an extrema only after traveling a certain distance past the last extrema. Similarly, a particle located at the correct position estimate will also be situated a little ahead of the feature associated with this particle. The incorporation of distance-matching ensures that there is no growing error in the position estimate. If not included, the position estimate would locate the position of the correct feature, but the location relative to the feature would be wrong since the vehicle has to drive past a feature before it figures out that a feature was detected.

The weight based on distance-match is also a Gaussian weighting function of the form in equation (4), as shown in Equation (8).

$$\mathrm{qd_i} = \mu^{-1} \cdot \exp\left(-0.5 \cdot \mathrm{K} \cdot {\mathrm{R_d}}^{-1} \cdot (\mathrm{diff(i)})^2\right) \tag{8}$$

with

$$\mathrm{diff(i)} = \|\mathrm{dist} - \mathrm{d(i)}\|_2 \tag{9}$$

where *d(i)* is the absolute distance between the ith particle and the feature prior to its position, *d* is the absolute distance between the feature just detected in the online phase and current position of the vehicle and *K* is the tuning factor. In place of R_θ in equation (3), R_d, the measurement noise variance in distance, is used. The measurement noise variance in distance is calculated in terms of measurement noise variance in odometry as

$$R_d = \sqrt{N}(R_\theta) \tag{10}$$

where $N = abs(P_v - P_F)/0.5$;

P_v is odometry value of vehicle at that time step, P_F is odometry value of measured feature, 0.5 is the map decimation. A tuning factor (K) was used in the weighting scheme and in the motion model of the particles. This was to vary the weight given to each particle based on the feature match as well as spread the particles more in the motion model so as to give them a higher chance of converging to the correct feature. Numerous tests were conducted to determine the value for this tuning factor, which are elaborated on in Section 5.4. A test was conducted on Dataset 1 to calculate how many pitch values are spanned on an average by a feature. It was found that for Dataset 1 an average of 92 pitch values were spanned by a feature. Hence for this work a value of 100 was used so as to also increase the spreading of particles after every re-sampling step.

The two weights are then combined as a weighted ratio to give a weight to each particle. Each of the two weights is normalized as follows

$$Npf_i = qf_i / \sum_i qf_i \tag{11}$$

where Npf_i is normalized weight of ith particle based on feature-match, qf_i is weight given to ith particle from Equation (5). Similarly,

$$Npd_i = qd_i / \sum_i qd_i \tag{12}$$

where Npd_i is normalized weight of ith particle based on distance-match, qd_i is weight given to ith particle from equation (8).

To combine the two weighting schemes a weighted average is used. The weighting scheme used is as follows:

$$p_i^k = 0.8 * \left(Npf_i^k\right) + 0.2 * \left(Npd_i^k\right) \tag{13}$$

where p_i^k is the weight given to ith particle at the kth time step, Npf_i^k is normalized weight based on feature matching from Equation (11) and Npd_i^k is normalized weight based on distance matching from equation (12). The ratio to combine the two weights was decided upon by a trial and error method and was found to give optimal results when the ratio of feature-matching weight to distance-matching weight was such that more preference was given to the feature-match. This ratio makes sense because agreement between feature vectors holds more importance than agreement in distance as the probability of getting a good distance-match even with the wrong particle is higher than a wrong feature-match. When a particle has a high feature-match it will have

a higher probability of being the correct feature. Figure 8 shows the weighting mechanism used for the particles.

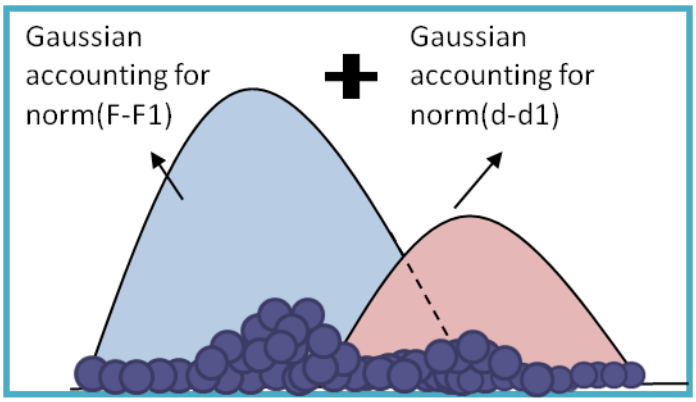

Figure 8: Weighting mechanism for particles

One can observe that the weighting scheme can be made more efficient by using a Gaussian weighting function every time a feature is detected. The variance value for the function can be calculated based on a Kalman filtering procedure to estimate variance based on values of feature and distance measurement variance for every feature detected. This method could be more efficient than the weighted average used here and is clearly an area for future work in this research.

The proposed method of using a weighted average to calculate weights of the particles has some disadvantages. There are two possible extreme scenarios that can be assumed. One, that the vehicle drives for a long time on the roadway and only encounters a unique feature after a long distance. In this case, the variance of the odometry will be very large due to the long distance traveled, while the variance of the feature will be extremely small. Taking the weighted average that gives more importance to feature-matching seems appropriate in this case. In contrast, in the scenario where the vehicle travels a very small distance and finds a less unique feature, the variance of the odometry will be small while that of the feature will be large. Now taking the

proposed weighted average will give a large number of particles a high weight. This seems unnecessary and would slow down the algorithm convergence. Hence the method used here, an 80/20 weighting, is a compromise that is seen to work well for roadway data. However it is clearly not the most optimal weighting scheme and further work can be done to improve on this aspect.

After this, the weights of the particles are normalized and then the re-sampling step is performed. Here the particles with a high weight are multiplied while the ones with a low weight are removed. This follows from the re-sampling algorithm described in [14] which is

c=cumsum(q^k)

u_1=rand(1).N^{-1}

i=1

for j=1…N

$u_j=u_1+(j-1).N^{-1}$

while $u_j>c_i$

i=i+1

end

$X_{p,j}^{k}=X_{p,i}^{k}$

end

where rand(1) is an evenly distributed random number in [0,1], N is the number of particles and cumsum is the cumulative sum such that c is calculated as:

$$c_i = \sum_{m=1}^{i} q_m^k \tag{14}$$

A particle with a weight less than N^{-1} is likely to be removed while a particle with a weight greater than N^{-1} stays, or is multiplied.

After every iteration the mean of the positions of all the particles is considered to be the position estimate. This procedure is carried on and eventually results in the effective localization and tracking of the vehicle.

The proposed algorithm was implemented on a dataset 11 km long measured from Interstate I-80 in Pennsylvania (Dataset 1). The results can be seen in Figure 9 below, which shows the position estimate error versus distance traveled. As can be seen, initially the position estimate error is very high since the particles are initially randomly distributed across the map and after

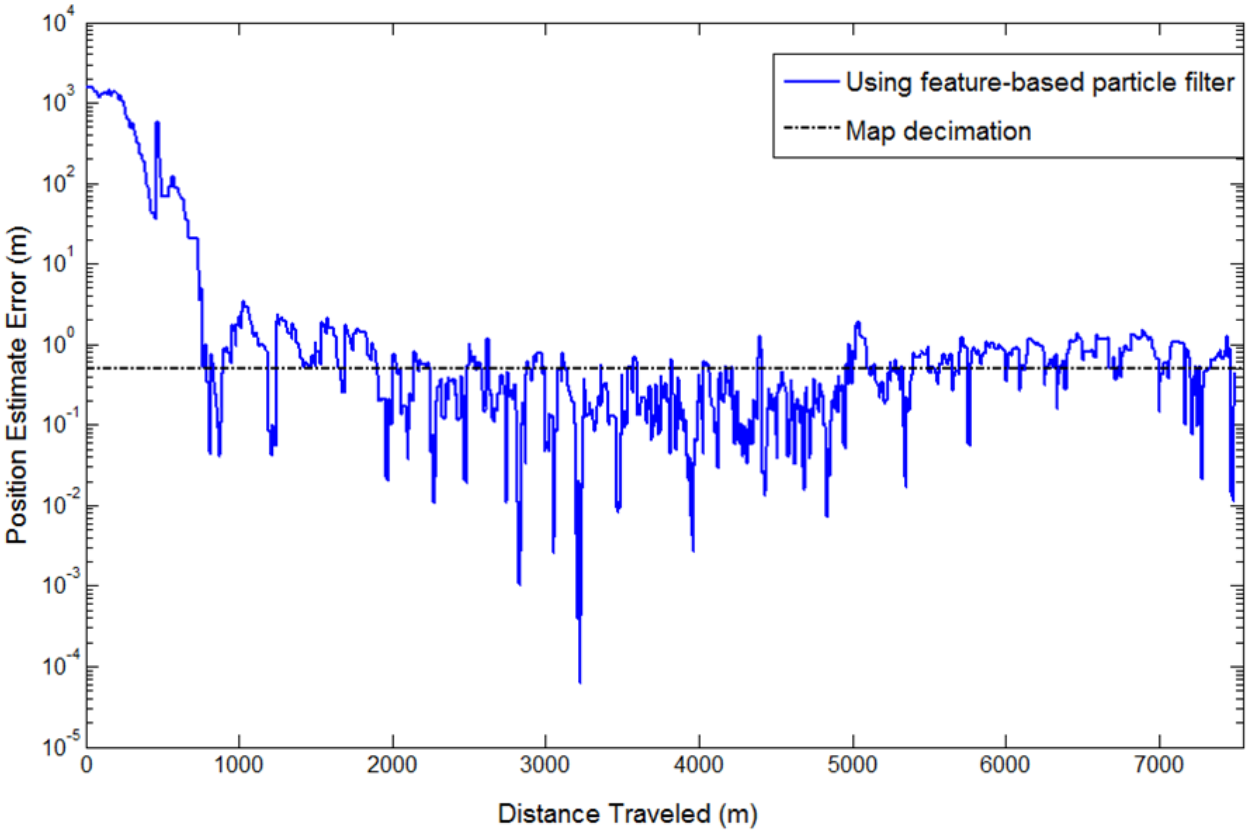

Figure 9: Prediction error vs distance traveled for Dataset 1

traveling about 1000 m error converges to a value of ≈ 0.5 m. The decimation of the map, that is, the rate of collection of pitch values is 0.5 m. Hence, the best a localization and tracking algorithm

could perform would be 0.5 m. The error after convergence, while tracking the vehicle, is mainly due to odometry sensor errors. This result is illustrative that the proposed method works. Tests on more datasets and a more detailed analysis of the results as well as a comparison of this algorithm with the classical Particle filter are performed in the next chapter.

CHAPTER 5
RESULTS

The Feature-based Particle filtering algorithm was tested on several datasets of pitch measurements taken from Interstate I-80 and US Route 220 in Pennsylvania. The results obtained were compared with those in [14], where pitch values were directly used for localization using a classical Particle filter without constructing features. Figure 10 shows a comparison of position estimate error versus distance traveled for both algorithms when implemented on Dataset 1 collected from Interstate I-80 for a distance of 11 km. The following subsections discuss convergence rates, accuracy and database requirements of the feature-based approach as compared to the classical Particle filter for Dataset 1 with reference to Figure 10.

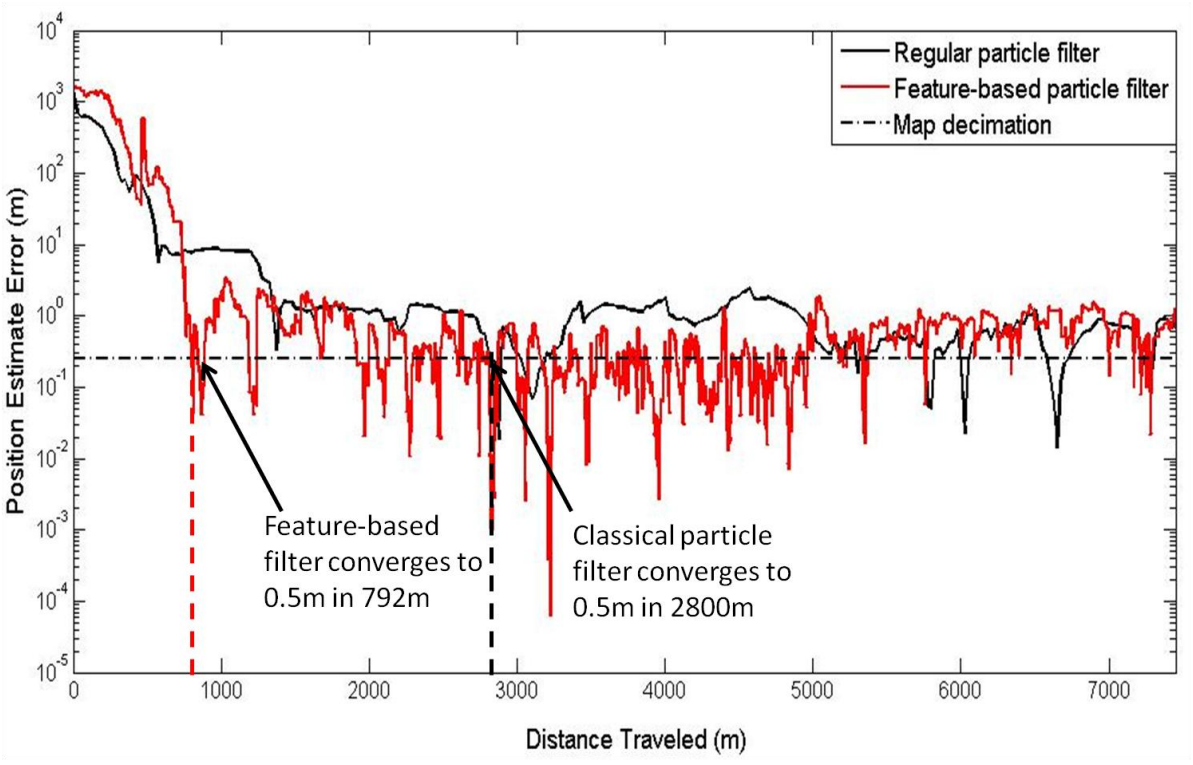

Figure 10: Prediction error vs distance traveled for Dataset 1 for classical and Feature-based Particle filter

The algorithm was run on the same dataset a 100 times and the distribution of average prediction error for them after convergence is shown in Figure 11.

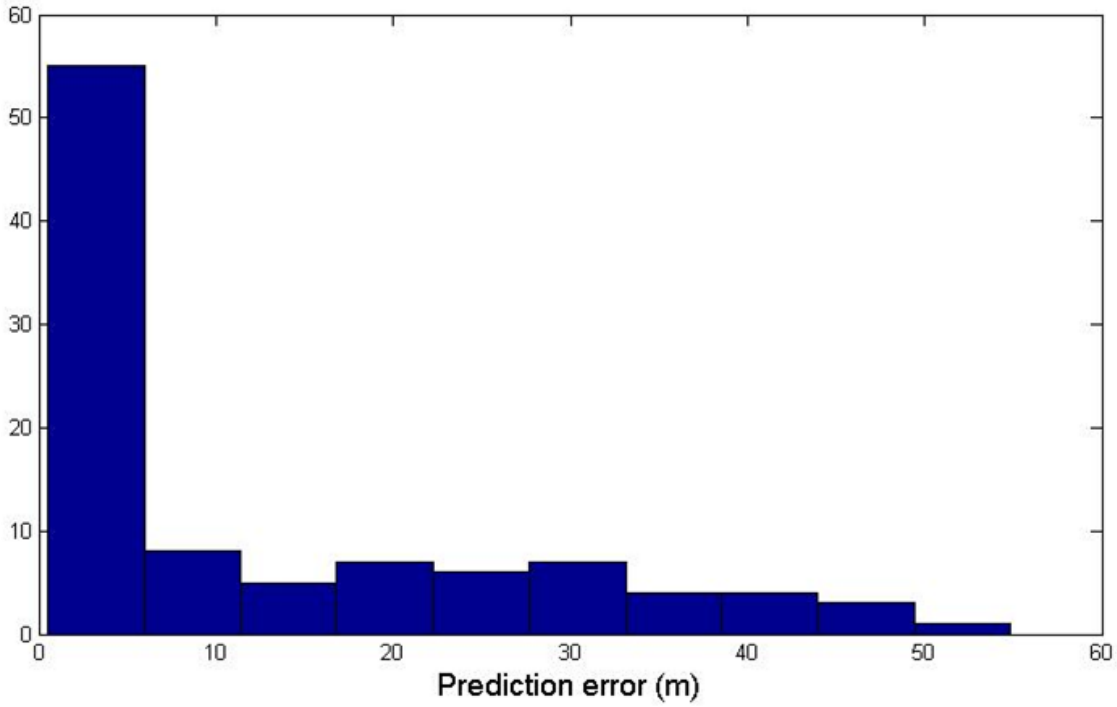

Figure 11: Histogram of mean of position estimate error when algorithm run on Dataset 1 a 100 times

5.1 Convergence rates and accuracy

In the current discussion, the convergence rate of a Particle filter is defined as the distance traveled by the vehicle for the filter to first obtain a position estimate error of 0.5 m. This was the definition chosen for convergence because the decimation of the map for the datasets used is 0.5 m, that is, pitch data was collected at the rate of 1 pitch value every 0.5 m of roadway. Hence the best any estimation procedure can do, given these datasets, in localizing and tracking a vehicle is 0.5 m. In Figure 10, the classical Particle filter converged to a position estimate error of less than 0.5 m after 2800 m of travel. On the other hand, the Feature-based Particle filter converged to a position estimate error of less than 0.5 m after only 792 m of travel. Thus, in this example, the error

in feature-based approach converges about four times faster than the classical Particle filter. Further, the average error after convergence in the case of the classical Particle filter was 0.7565m, while for the Feature-based Particle filter the average prediction error was 0.5984m.

For this simulation, the number of particles placed on the map for the classical Particle filtering approach was 1000 particles/mile (as explained in [14]). The Feature-based Particle filter gave a better accuracy and a faster convergence with just 250 particles/mile, one-fourth the number in the case of the classical Particle filter. This greatly reduces the computational effort required for the feature-based approach.

Table 1 shows that there is 10 times less computational effort involved for the Feature-based filter as compared to the classical Particle filter. This is further elaborated on in Section 5.3. Thus there is a 10x improvement in terms of computational effort given the same number of particles and a 4x improvement in terms of number of particles, giving an overall 40x improvement on the existing Particle filtering technique for this dataset.

A similar comparison test was done for a Dataset 2. The results are shown in Figure 12. As can be seen, here too the Feature-based Particle filter had a better accuracy and a faster convergence rate. In the case of the classical Particle filter, the position estimate error converged to a value below 0.5 m after 2625 m of distance traveled. The Feature-based Particle filter converged to a position estimate error of below 0.5 m after 1321 m of distance traveled. Hence the convergence rate of the feature-based approach was nearly twice that of the classical Particle filter. Also, the average position estimate error in the case of the classical Particle filter was 1.84 m while that of the Feature-based Particle filter was 0.84 m. In this simulation, the classical Particle filter again used 1000 particles/mile while the Feature-based Particle filter gave a better accuracy and faster convergence using just 500 particles/mile, half the number of particles used by the classical

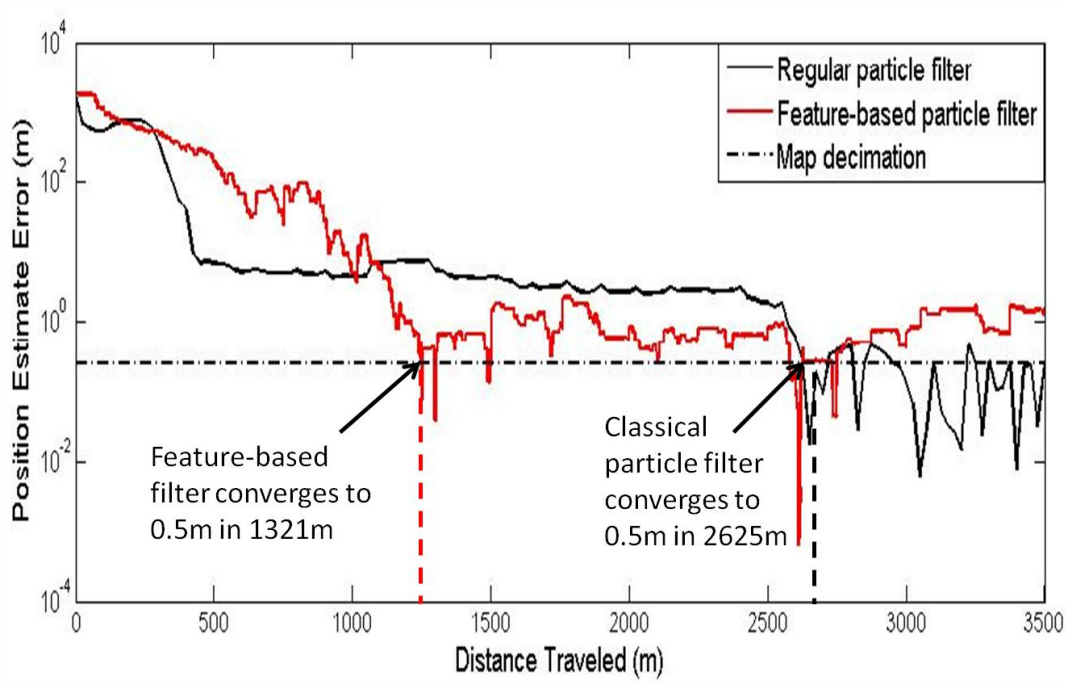

Figure 12: Prediction error vs distance traveled for Dataset 2

Particle filter. Hence in this case, there was an overall improvement of 20x as compared to the classical Particle filter.

5.2 Database size

Another important aspect of comparison is the database of information that needs to be stored in both cases. The Feature-based Particle filter needs to store information pertaining only to the feature vectors, whereas the classical Particle filter needs to store all the pitch values collected from the roadway. Thus, as expected, the database required is much smaller than for the classical Particle filter. For simulation using Dataset 1, the storage capacity required for the classical Particle filter was 4.10 MB, while the storage capacity required for the Feature-based Particle filter was only 55.2 KB, a reduction of a factor of 75. In the case of Dataset 2, the classical Particle filter needed a

database capacity of 2.46 MB while the Feature-based filter needed a database capacity of 6.63 KB, an improvement by a factor of 380.

5.3 Computational effort

An important aspect of comparison between the two algorithms being analyzed is the computational effort involved. The classical Particle filtering algorithm performs the update and re-sampling functions at regular time steps (after every 25 m of travel in [14]). Hence it was expected that the computational effort of this algorithm would be much higher than that of the Feature-based Particle filter, which performs the re-sampling step only when a feature is detected. A comparison of the number of computations involved in both algorithms when only one particle goes through the entire process for all time steps was performed for Dataset 1. As was expected, the number of FLOP counts in the classical Particle filter was 3.06E8 while that in the Feature-based Particle filter was 3.52E7. The details of these calculations are described in Table 1. The values for FLOP counts used were referenced from [39]. As can be seen, there is an order of magnitude difference between the computational effort involved for the two algorithms, with the feature-based method being less computationally intensive. The decrease in computational effort in the feature-based approach provides a significant advantage over the classical Particle filter.

The discussed results indicate that the Feature-based Particle filter is more accurate, has a faster convergence rate and needs less computational effort. This makes the feature-based approach more practical in terms of real-time implementation and shows that the Feature-based filter successfully optimizes the classical Particle filtering approach used previously for this application.

Table 1: Computational Effort Comparison

OPERATION	FEATURE-BASED APPROACH	CLASSICAL PARTICLE FILTER
ADDITIONS	20,601,639	153,020,000
SUBTRACTIONS	825,293	4,003
MULTIPLICATIONS	22,983	6,000
DIVISIONS	3,447,375	38,260,000
EXPONENTIATIONS	2,151	2,000
TOTAL FLOP COUNTS	35,256,623	306,086,003

5.4 Parameters of the Feature-based algorithm

The number of particles used for the simulations and details of the weighting mechanism used are described in detail below.

(1) **Particle Population size:**

As described in [14], for the initial Particle filter implementation, the number of particles was set to 1000 particles/mile. This number was reached upon by performing numerous simulations on different datasets. In the current work it was anticipated that the number of particles required would be fewer than the classical Particle filter implementation due to the fact that features are more unique than pitch values. The number of particles in a classical Particle filter needs to be kept high so as to cater for errors in position estimation via pitch values, to avoid the convergence of all the particles to a wrong position estimate and to increase the robustness of the algorithm. Thus, a number of tests were carried out where the number of particles used was scaled down.

In the case of Dataset 1, it was found that the Feature-based Particle filter performed better than the classical Particle filtering approach even when the particle population size was reduced to one-fourth the initial number, that is, 250 particles/mile of roadway. Figure 13 shows the position estimate error versus distance traveled for different particle population sizes for Dataset 1. It can be seen that initially when the number of particles is reduced to half and then one-fourth, the algorithm performs better, but when it is reduced further on, to say one-eighth, the algorithm does not converge at all. Hence, there is a certain 'sweet spot' while deciding on the number of particles. Figure 14 shows a plot of the mean error value after convergence versus the number of particles used, for Dataset 1. As can be seen, beyond the value of 250 particles/mile the

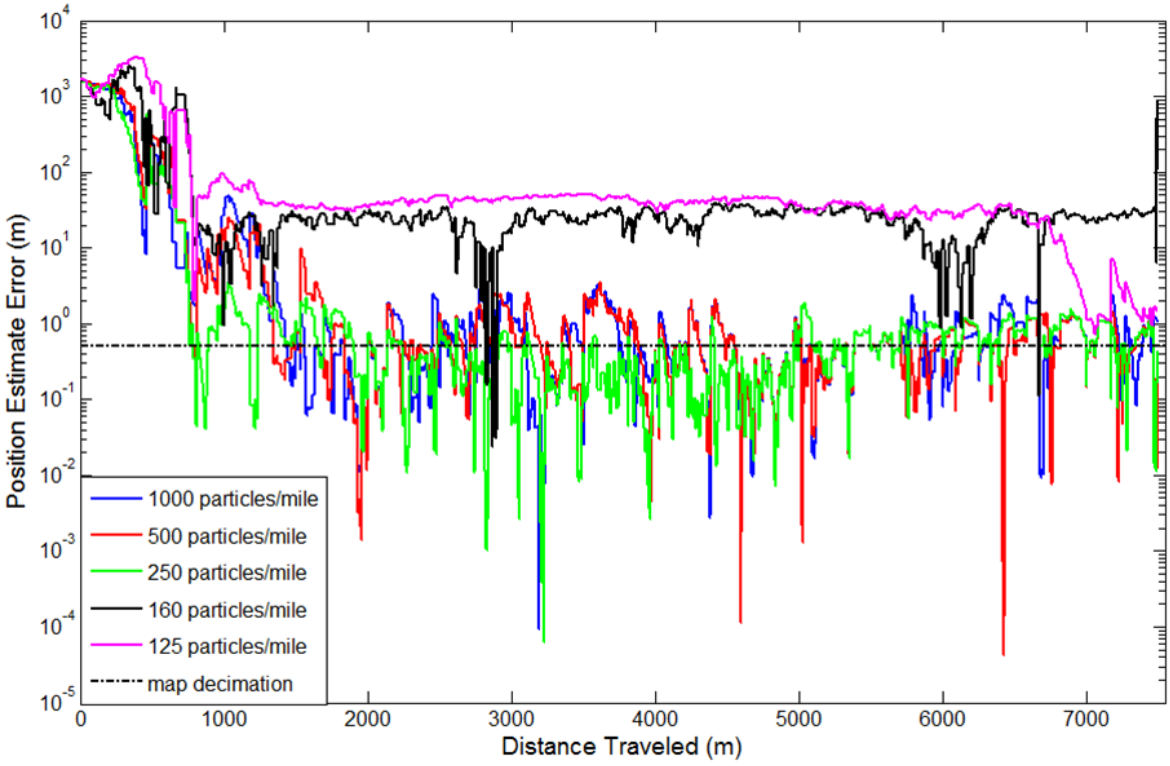

Figure 13: Position estimate error vs distance traveled for Dataset 1 with varying particle population size

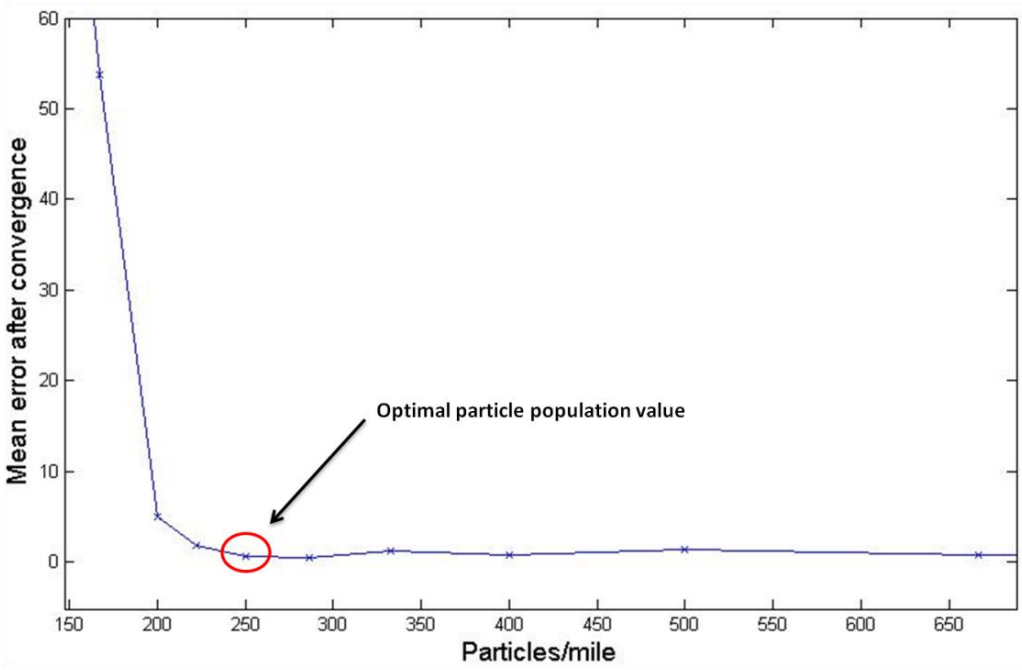

Figure 14: Mean error after convergence vs particle population size for Dataset 1

change in mean error value is not very significant. Hence the optimal particle population size for this dataset is 250 particles/mile. If the number is made too low, the algorithm is starved of particles and the likelihood of converging to the wrong position increases. Either way, the number of particles required is found to always be lesser than that for the classical Particle filtering case, hence significantly reducing the computational burden.

Figure 15 shows the same tests carried out for Dataset 2. Here the optimal population size was found to be 500 particles/mile. In Figure 16 it can been seen that if the number of particles is increased beyond 500 particles/mile the improvement in mean error after convergence is not significant. Hence the choice of 500 particles/mile is beneficial in the case of this dataset.

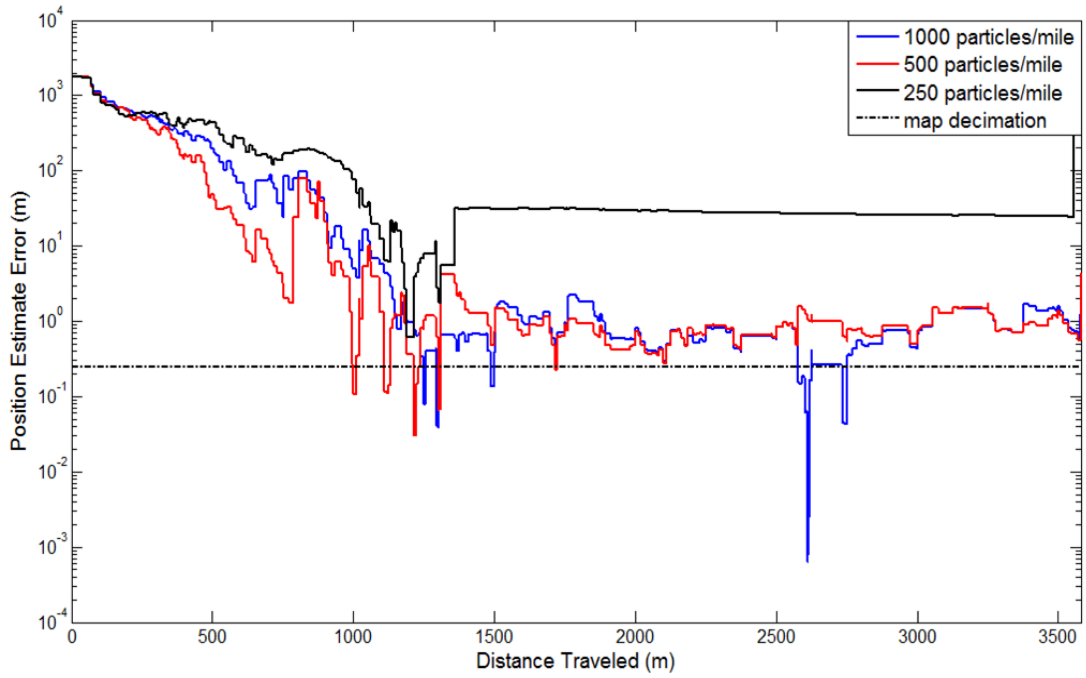

Figure 15: Position estimate error vs distance traveled for Dataset 2 with varying particle population size

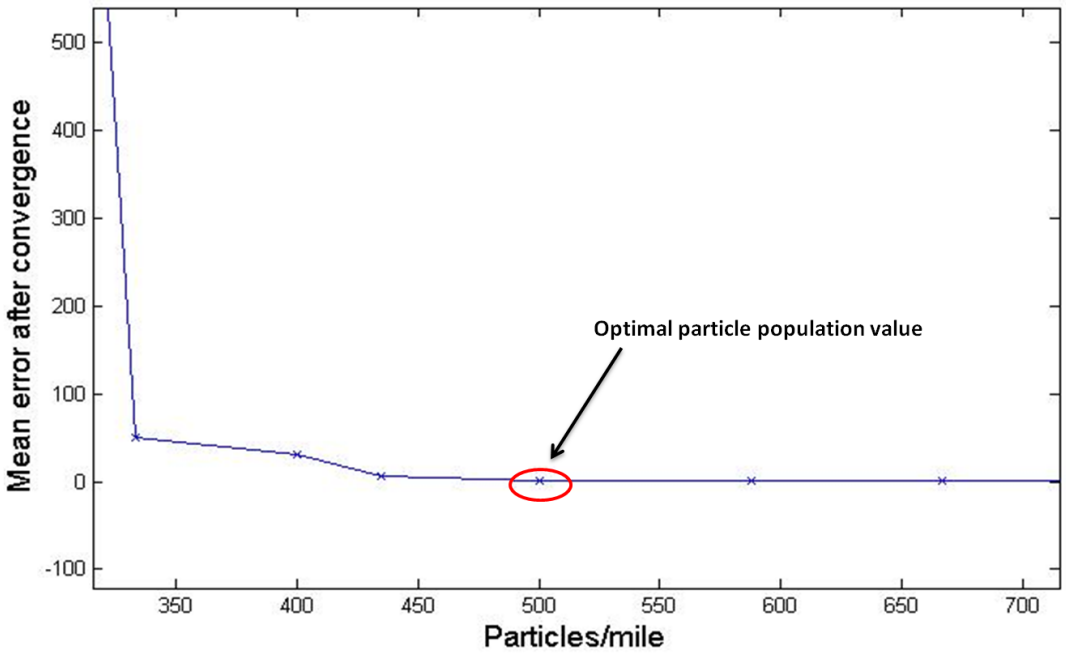

Figure 16: Mean error after convergence vs particle population size for Dataset 2

(2) **Choice of particle weighting function**:

The weights given to the particles after a feature is detected in the online phase should be given based on the extent of feature-matching and distance-matching for each individual particle. But the aspect of noise and possible errors should also be kept in mind. The particles along the map are weighted according to their agreement with the feature detected online using a Gaussian weighting function.

As described in [14], the Gaussian weighting function is of the form

$$q_i = \eta^{-1}.\exp\left(-0.5.R^{-1}.diff^2\right) \tag{15}$$

where q_i is weight of the i^{th} particle, η is a normalizing factor equal to the sum of the particle weights, R is the variance in noise and *diff* is the difference between the relevant measurement and the particle's corresponding value for that measurement from the map.

To decide on the weighting scheme the error in feature detection was observed. For Dataset 1 the feature that should have been detected at each time step was noted and the feature that was detected in real-time was noted. The error in feature detection had a distribution as shown in Figure 17. Since the error measure used here was the 2-norm, all values obtained were positive. This distribution could be modeled as a half-normal distribution and hence for this work a Gaussian weighting scheme was used.

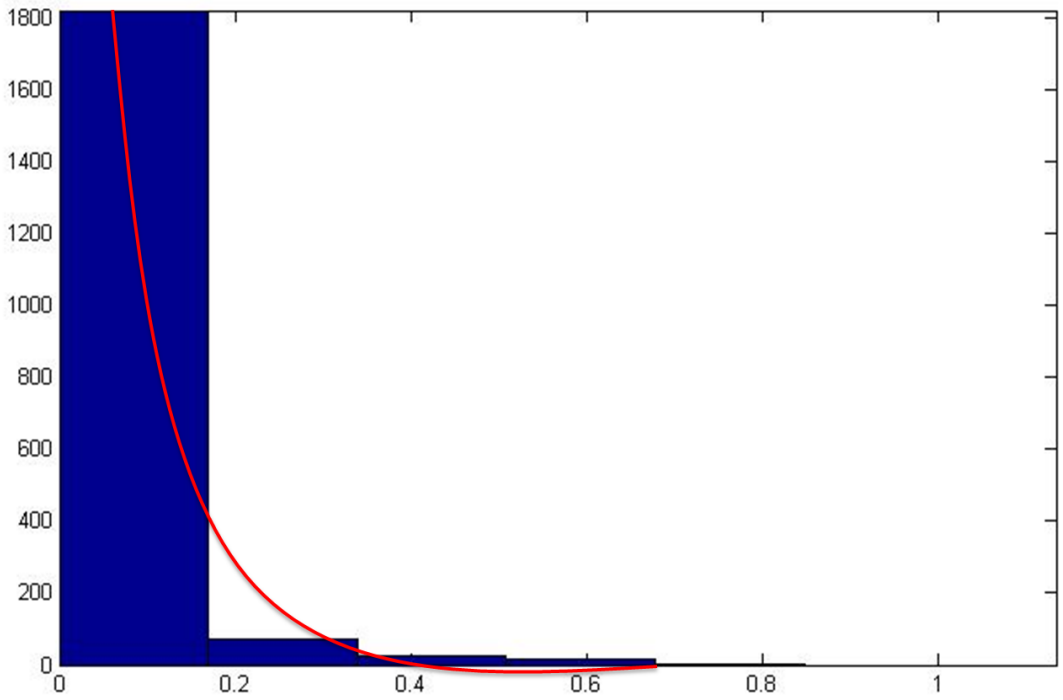

Figure 17: Histogram of error in feature detection for Dataset 1.

Gaussian weighting function ensures that the particle that could correspond to the actual position is not eliminated in the re-sampling step due an error in measurement or measurement noise. For this to happen, the tail of the Gaussian weighting function should be long enough to keep the correct particle in the map. We now have two Gaussian weighting functions, one based on feature-matching and the other based on distance-matching. It was decided to combine the two using a weighted average. This makes sense because we would like a particle that gets a high feature-match and a high distance-match to get a high weight whereas a particle that gets only a good feature-match or only a good distance-match is probably not at the correct position and hence should get a lower weight. These conditions are accommodated in the weighted sum approach. The ratio to combine the two weights was decided upon by a trial and error method and was found to give optimal results when the ratio of feature-matching weight to the distance-matching weight was 80/20. This ratio makes sense because a feature-match holds more

importance than a distance-match due to the uniqueness associated with a feature. The probability of getting a good distance-match even with the wrong particle is more than a wrong feature-match. Hence, when a particle has a high feature-match it will have a higher probability of being close to the correct position.

(3) C**hoice of tuning factor**:

The tuning factor used in this work was incorporated because the distance traveled between detection of features varies, sometimes being very long. A test was run on Dataset 1 to determine the average number of pitch values spanned by a feature and was found to be 92. Hence a tuning factor value of 100 was used in this work to account for this fact. The tuning factor was also incorporated in the motion model of the particles so as to spread them more after each re-sampling step, hence giving them a higher chance of converging to the correct feature. Another aspect to be considered is that giving a very good feature match a high weight in the weighting scheme would lead to a higher convergence rate, but would also increase the probability of converging to an incorrect feature at the start. Hence there is a compromise to be made. As can be seen in Figure 18, when a low value of K (below 90) is used, the filter does not converge at all. If we do not give a good feature match a high weight, the filter may never converge or would require a long time to do so. On the other hand, if a very large value of K (above 130) is used, then the filter does not converge as now too high a weight is given to the wrong feature early on. Tests conducted suggest that a tuning factor in the range of 90 to 130 works well. Optimal results were obtained in the case of the datasets used when the tuning factor was set to 100. The sensitivity of the algorithm to K is an area of future work.

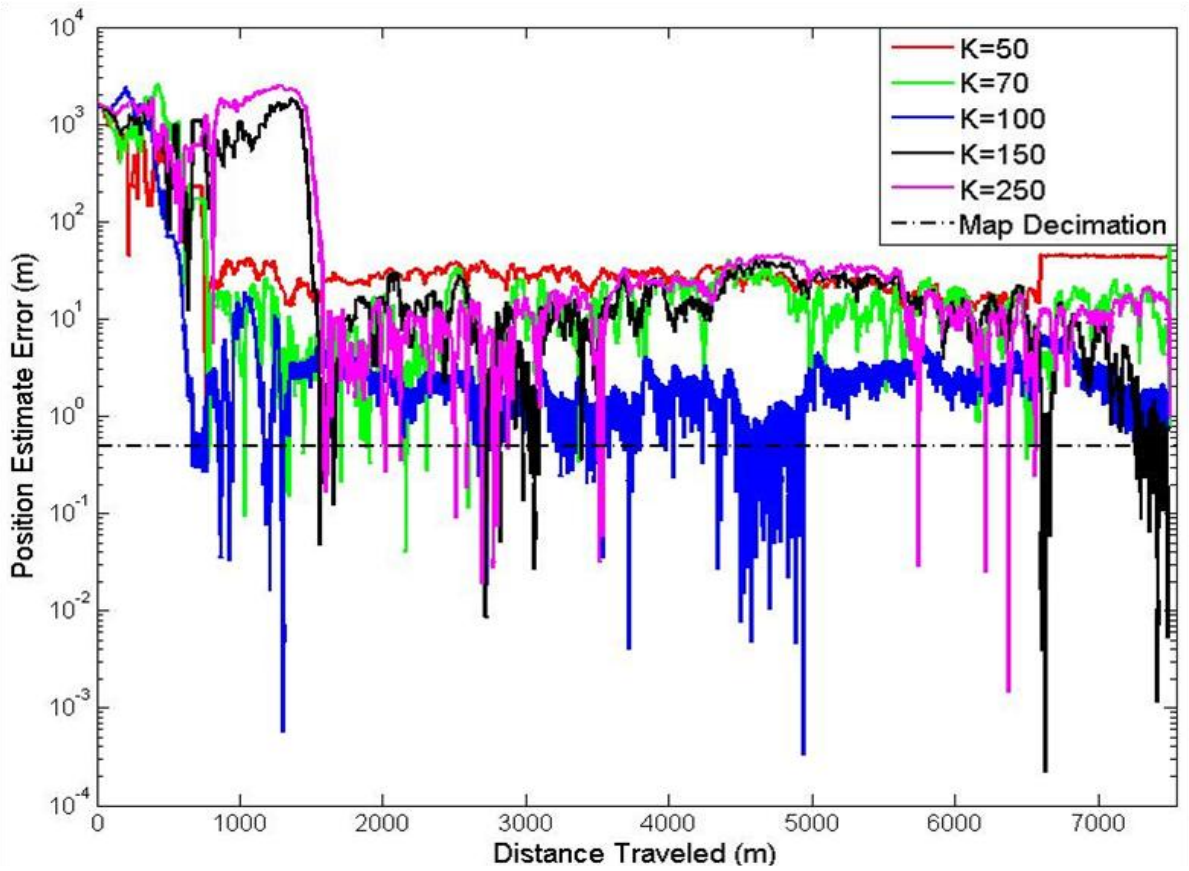

Figure 18: Position estimate error vs distance traveled for Dataset 1 with different values of tuning factor (K)

5.5 Tests conducted

To ensure that the algorithm was in fact getting latched on to the correct feature and that the highest weight was in fact being given to the correct match, the following test was conducted. First an algorithm was run so as to identify the features that should be detected as the vehicle travels down the roadway and their order of detection. These set of features were then stored as a database and given numbers in ascending order based on order of detection. Then the Feature-based Particle filter was run and the feature latched onto as the correct feature was recorded at each time step. These two sets of features were later compared. Figure 19 shows the results obtained when the test was run for Dataset 1. Here the ordinate is the numerical difference between the numbers of the feature that should have been detected and the one that was latched

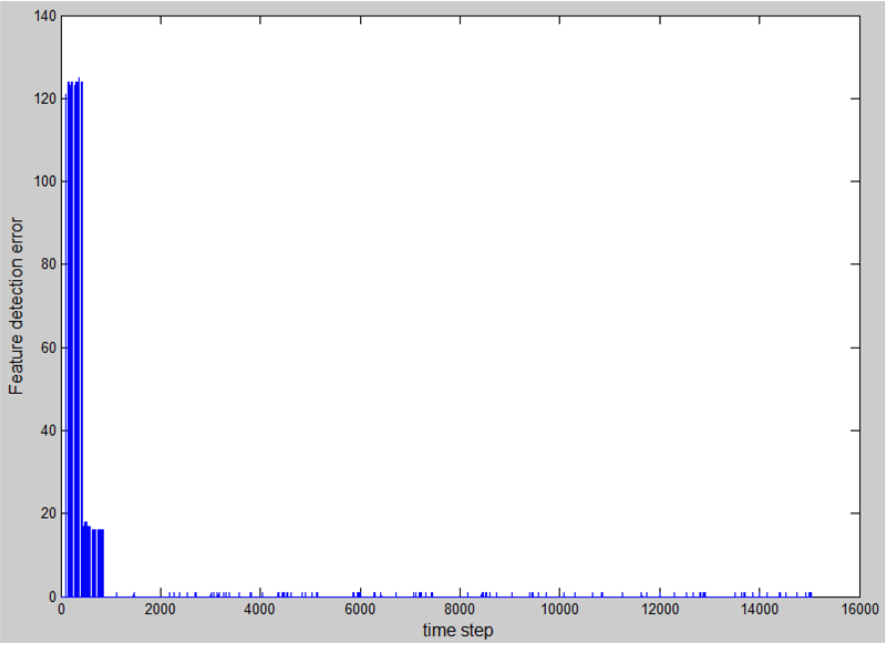

Figure 19: Feature detection error vs feature number for Dataset 1

onto. Figure 20 shows an enlarged view of Figure 19. Here it can be seen that initially the feature

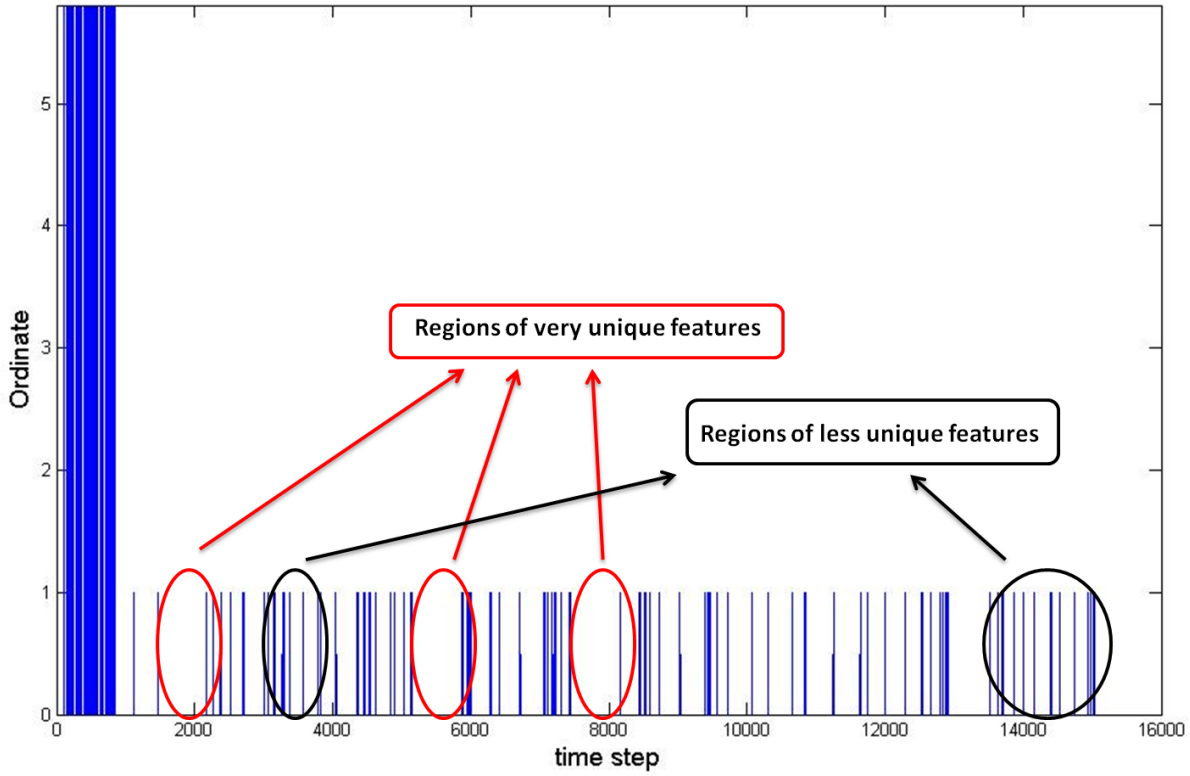

Figure 20: Enlarged view of Figure 19

latched onto was wrong and hence the ordinate value was very high. As the time steps increase this value decreases and then varies between 1 and 0. A value of 1 indicates that the particles are latching onto features close to the correct feature but not the correct one. A value of 0 indicates that the correct feature has been latched onto. A clustering effect can also be observed in Figure 20. The regions where clusters of value 1 are observed would be expected to be regions of less unique features while the regions with a value of 0 would have highly unique features.

Figures 21 and 22 show the same results for the test conducted on Dataset 2. Again we observe a high value of ordinate in the initial time steps after which the value changes between 0 and 2.

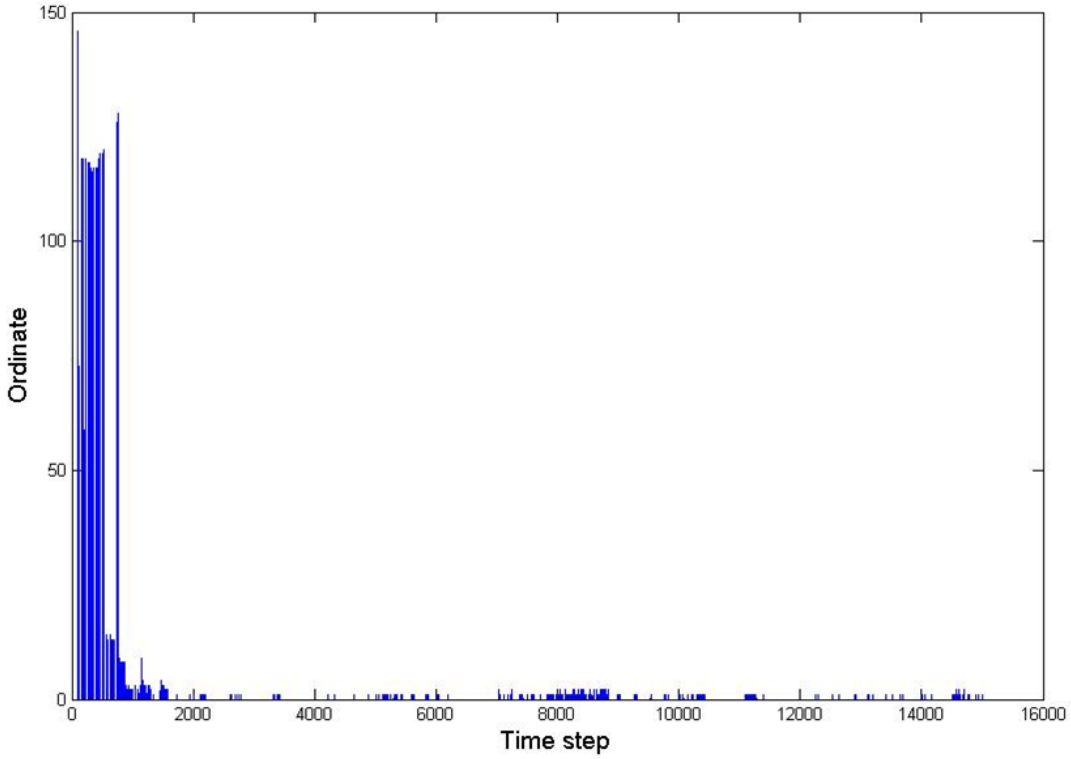

Figure 21: Feature detection error vs feature number for Dataset 2

Hence we observe that wrong features are latched onto even after convergence if the features are onto even after convergence if the features are not very unique. But the correct feature can be found the moment a unique feature is detected.

The current set up can localize and track the vehicle with sub-meter accuracies and is an improvement over the existing classical Particle filtering approach. Yet results obtained indicate that there are areas that can be worked on to improve the accuracy and robustness of the Feature-based Particle filter. The next section describes the future work possible in this area and also mentions the different possible applications of this research.

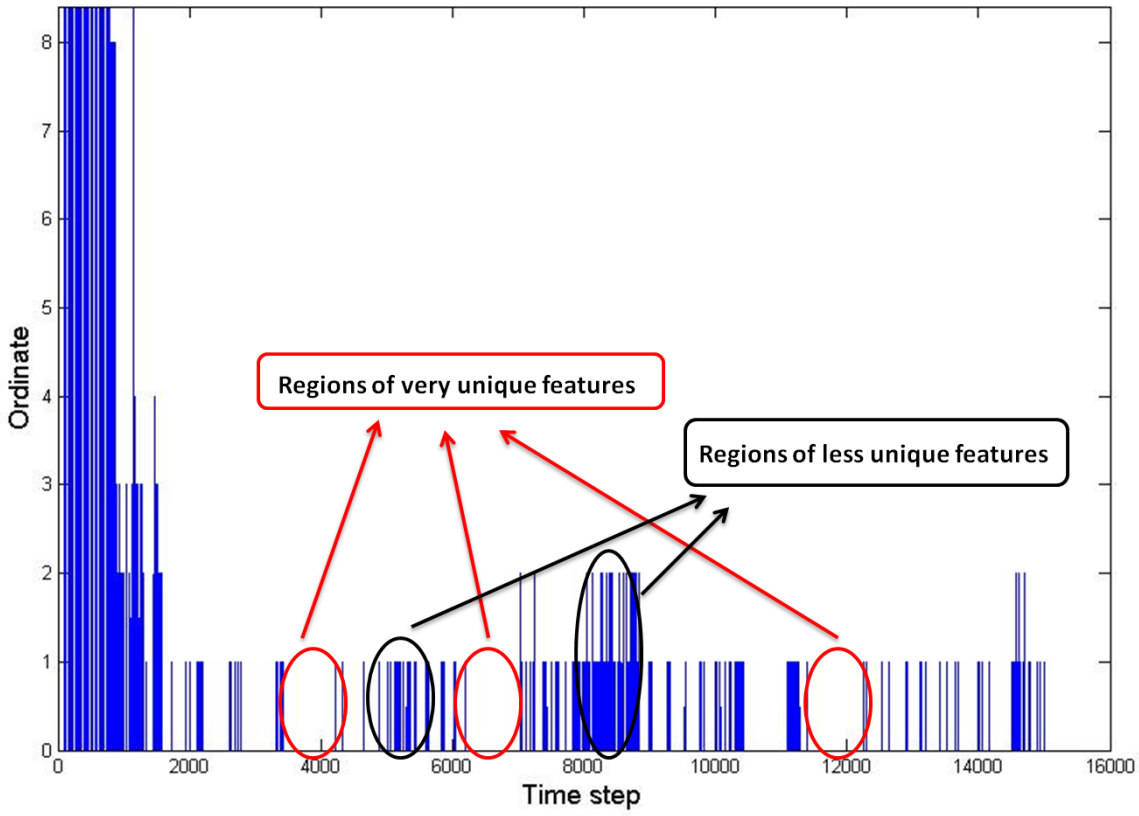

Figure 22: Enlarged view of Figure 21

CHAPTER 6
APPLICATIONS AND FUTURE WORK

This chapter describes the scope for future work in this research area as well as mentions its varied possible applications to domains other than the problem tackled in this thesis. The last section talks about the limitations of the proposed approach.

6.1 Future work

There is scope for future work in this domain with an aim of improving on the proposed algorithm. Further studies can be conducted wherein Feature-based Particle filtering is delved into deeper. Specifically, the weighting mechanism chosen in this work is a regular weighted average of the weights based on feature-matching and distance-matching. The present work uses a tuning factor (K) whose value was obtained via tests. In the future an in-depth study can be done to obtain a weighting mechanism wherein the variance value is calculated every time the vehicle detects a feature, using a Kalman filtering approach. This would account for both the extreme cases: of detecting a less unique feature often, hence having a low variance in odometry noise but a high variance in feature noise, and of detecting very unique features after long distances of travel, hence having low variance in feature noise but a high variance in odometry measurement. The Kalman Filter approach can use both the variance values each time a feature is detected and calculate the variance value to be used in the Feature-based Particle filter taking their values into account. Also, data storage in the map can be optimized to improve efficiency and reduce storage space used. Another area that can be looked into is a KD tree [26] or vocabulary tree-based [27] storage technique.

An alternate approach for vehicle localization and tracking would be to run the Particle filter by itself with the feature detection algorithm in parallel. Whenever the Particle filter looses convergence, the global position obtained via the feature detection algorithm can be used to re-initialize the filter. Meanwhile, bias and scale error can be calculated by the feature algorithm and supplied to the Particle filter being used for state estimation to improve on its accuracy.

Another interesting approach could be to solve the localization and tracking problem using a systems approach wherein the entire roadway is modeled as different plants in the preprocessing phase. As the vehicle travels down the road and collects pitch information, a test can be performed to know which plant model it is best-suited to and hence converge to the correct location. A similar idea has been worked on in [42], where Ozay, Sznaier and Lagoa discuss the problem of robust identification of hybrid systems in a set membership framework.

And lastly, a real-time implementation of the algorithm can be performed. Previous work by A. Dean [14] performed a real-time implementation of the filtering algorithm using raw pitch values. The present work makes use of features which are more unique and less memory intensive than previous work and hence a real-time implementation is feasible. Hence there are a number of areas in which this research can be built upon.

6.2 Applications

There can be numerous applications for the work described here. The application aimed at here was the localization and tracking of a vehicle. Hence we used terrain data (pitch data) to generate features. Specifically for this application, roll data or a combination of roll and pitch data can also be used. Other applications include audio track retrieval [21][40][41], where the information from music can be used to form unique features. Similarly, the method can also be

used in the Stock market, to obtain unique features (trends) in the change in stock values. The algorithm can then be used as an estimator of future behavior of the stock. Hence this method can be used in varied applications for problems pertaining to live streaming data, their analysis and retrieval of information (features) that can describe the data uniquely and possibly predict or estimate its behavior in the future. Hence, this work has relevance not only in the field of vehicle position estimation but also in numerous other areas.

6.3 Limitations

One scenario in which the classical Particle filter will work better than the Feature-based filter is on very short roadways. This is because on shorter roadways the probability of finding unique features is smaller, while the classical Particle filter will continue to perform in the same manner even on very short roadways. If the roadway happens to have a significant amount of grade change at short distances, then the Feature-based filter will still work well.

6.4 Conclusion

This thesis proposed a new technique to efficiently localize and track a vehicle using the concepts of feature extraction and Particle filtering. The Feature-based Particle filter was found to successfully localize and track a vehicle using the datasets collected from Interstate I-80 in Pennsylvania. This technique has many areas that can be worked on, which include the weighting mechanism for the particles in the algorithm. It was found to be a great improvement (40 times

improvement for Dataset 1 and 20 times improvement for Dataset 2) over the classical Particle filter in terms of convergence rates, memory requirements and computational effort. There are many areas for improvement but overall this algorithm is just a first step towards the trend of combining techniques used for Global localization and local tracking.

Appendix 1: MATLAB code for the Feature-based Particle filter algorithm

```
%-------------------------------------------------------------------------
%     Feature-based Particle Filter Algorithm
%     Code written by:
%     Sneha Kadetotad
%-------------------------------------------------------------------------
clc
clear all
close all

% Initialize the parameters necessary for preparing the map data:
par.N      = length(INSPVAData(1,:));        % Number of time steps
par.ffilt  = 0.1;                            % Cutoff frequency for low
                                               pass filter (cycles/meter)
par.travel = 0.5;                            % Distance traveled between
                                               iterations (m)
par.dx     = 0.5;                            % Map decimation (m)
temp.startpos  = 0;                          % Set the starting position
                                               (in % of the total length)

% Clip the maps (in % of the total length)
temp.mapstart  = 0.0;
temp.mapend    = 1.0;
temp.fragstart = 0.0;
temp.fragend   = 1.0;

%-------------------------------------------------------------------------
% Import the data:
Script_import_data
frag_dist=frag.odom(1,2:end)-frag.odom(1,1:end-1);
frag_dist=[0 frag_dist(1:15094)];

%-------------------------------------------------------------------------
% Prepare database of features from map data
% Import the feature database
load('database of features for map data');
feat_loc_in_map=map.odom(feat_loc);

%-------------------------------------------------------------------------
% Initialize the remaining PF parameters:
par.S =floor(floor(map.D(end)/0.621371192237)/4);   % Particles per time
                                                      step (1000/mile)
par.Rwx =100*(0.01*par.travel)^2;                   % Odometry measurement
                                                      variance
par.Rp = 100*0.1;                                   % Measurement noise
                                                      variance in pitch
```

```
par.Rr = 100*0.1;                                   % Measurement noise
                                                      variance in roll

%-------------------------------------------------------------------------
% Predict the steady state variance P:
map.pitch_corr = interp1(map.D, map.pitch, [0:par.travel:max(map.D)]);
map.dtheta_dx_mean = sqrt(mean((diff(map.pitch_corr)/par.travel).^2));

prediction.P =
par.Rwx/2*(1+sqrt(1+4*par.Rp/map.dtheta_dx_mean^2/par.Rwx));

%-------------------------------------------------------------------------
% Initialize the particles, their positions, and their weights
orig_vec=zeros(500,par.S);
xu.long=orig_vec;
particle.long=orig_vec;
particle.PI_feat=orig_vec;
particle.PI_dist=orig_vec;
particle.PI=orig_vec;
particle.q=orig_vec;
particle.long(1,1:par.S) =rand(1,par.S)*(max(map.D)-
min(map.D))+min(map.D);
particle.q(1,1:par.S) = 1/par.S;

%-------------------------------------------------------------------------
% Preprocessing for the feature vector generation

% get the data
    i=0;
    load('452ParkLn2_frag');
    pitch_cell=INSPVAData(10,:);
    size_pitch_cell=max(size(pitch_cell))-1;

% set the initial settings
    enc_increment=0.05;
    settings_featvect.wv_sizemat=[3,1];
    settings_featvect.graphics_flag=0;
    settings_featvect.edge_flag=0;
    settings_featvect.enc_disp=enc_increment;
    settings_featvect.wavelet_min=0.02/settings_featvect.enc_disp;
    settings_featvect.wavelet_max=0.03/settings_featvect.enc_disp;
    settings_featvect.wavename='Gauss';
    settings_featvect.map_type='all_peaks';
    settings_featvect.feature_vect='differences';
    settings_featvect.name='d';
    settings_featvect.scale_factor=floor(log2(length(pitch_cell)));

% process to get the par thresholds
    full_settings_featvect=settings_featvect;
    [rwt_full,rwt_bit_full,tap_sum]=
RWT2(pitch_cell(1:2^settings_featvect.scale_factor),1,full_settings_featve
ct.wavename,1,2,full_settings_featvect);
    rwt_full=rwt_full*(2^(settings_featvect.scale_factor/2));

% get the par thresholds
```

```
    settings_featvect.par=std(rwt_full)/50;
    cell_filters = fcn_generate_wavelet_filters(settings_featvect);
    scfd=[];
    filter_level=1;

%-------------------------------------------------------------------------
ind=0;
v=1;
temp.div=1000;

% Start the iteration process
for index=1:par.N

    ind=ind+1;
    v=v+1;
    if(ind==500)
        v=1;
    end

    if(mod(ind,501)==0)
        xu.long_store=xu.long(ind-1,:);
        particle.long_store=particle.long(ind-1,:);
        particle.PI_store=particle.PI(ind-1,:);
        particle.q_store=particle.q(ind-1,:);

        xu.long(2:end,:)=orig_vec(1:499,:);
        particle.long(2:end,:)=orig_vec(1:499,:);
        particle.PI(2:end,:)=orig_vec(1:499,:);
        particle.q(2:end,:)=orig_vec(1:499,:);

        xu.long(1,:)=xu.long_store;
        particle.long(1,:)=particle.long_store;
        particle.PI(1,:)=particle.PI_store;
        particle.q(1,:)=particle.q_store;
    end


    if(ind==501)
        ind=1;
    end


% Move the postition of the particles according to the estimate and the
% measured odometry
xu.long(ind,:) = particle.long(ind,:) + frag_dist(index) +
sqrt(par.Rwx)*randn(1,par.S);


% Measure the particle's X and Y locations and corresponding pitch from
% the map using this script:
Script_particle_parameters_testrun
parx=particle.x;
pary=particle.y;
```

```
[scfd] =
fcn_ct_wavelet_filter(pitch_cell(index),scfd,cell_filters,settings_featvec
t);
update_flag(index)=scfd.fe_d{1,1}.uf;
if(update_flag(index)==1) % Feature detected
    for i=1:par.S
        distance = repmat(xu.long(ind,i),1074,1)-feat_loc_in_map' ;
        t=distance(find(distance>=0));
        if(isempty(t)==0)
           [value,junk]=min(t);
            pos(i)=find(distance==value);
            dist1=abs(frag.odom(index)-
                  frag.odom(scfd.fe_d{1,1}.feature_loc));

            n=ceil(dist1)/0.5;
            dist2=distance(pos(i));
            particle.PI_feat(v,i)=exp(-
              0.5/(par.Rp).*((norm(feat_vec(pos(i),1:5)-
              scfd.fe_d{1,1}.feature_vector(1:5)')).^2));
            particle.PI_dist(v,i)=exp((-0.5/(sqrt(n)*par.Rwx)).*(dist1-
              dist2).^2);
        else
            particle.PI_feat(v,i)=0;
            particle.PI_dist(v,i)=0;
        end
   end

% Normalize the weights

particle.PI_feat(v,:) = particle.PI_feat(v,:)./sum(particle.PI_feat(v,:));
particle.PI_dist(v,:) = particle.PI_dist(v,:)./sum(particle.PI_dist(v,:));
particle.PI(v,:) = (0.8).*particle.PI_feat(v,:) +
                                        (0.2).*particle.PI_dist(v,:);
particle.q(v,:)=particle.PI(v,:)./sum(particle.PI(v,:));


% Resample the particles according to the particles weight:
Script_SIR_testrun

else
particle.q(v,:)=particle.q(ind,:);
particle.long(v,:)=xu.long(ind,:);
end
end

Script_plot_results_testrun
```

Appendix 2: MATLAB code for importing data

```
%-------------------------------------------------------------------------
% Load map data

load('452ParkLn2_map.mat');

% Shift data to starting position
temp.startpos_i  = floor(temp.startpos*length(INSPVAData(2,:)));
if(temp.startpos_i == 0);
    temp.startpos_i = 1;
    temp.added_odom = 0;
else
    temp.added_odom = sqrt((INSPVAData(15,temp.startpos_i)-
(INSPVAData(15,temp.startpos_i-1)))^2 +...
                        (INSPVAData(16,temp.startpos_i)-
(INSPVAData(16,temp.startpos_i-1)))^2 +...
                        (INSPVAData(17,temp.startpos_i)-
(INSPVAData(17,temp.startpos_i-1)))^2);

% Correct the odom and D before shifting
INSPVAData(12,1:temp.startpos_i-1) = INSPVAData(12,1:temp.startpos_i-1) +
    INSPVAData(12,length(INSPVAData(2,:))) + temp.added_odom;
INSPVAData(12,:) = INSPVAData(12,:) - INSPVAData(12,temp.startpos_i);
INSPVAData(18,1:temp.startpos_i-1) = INSPVAData(18,1:temp.startpos_i-1) +
    INSPVAData(18,length(INSPVAData(2,:))) + temp.added_odom;
INSPVAData(18,:) = INSPVAData(18,:) - INSPVAData(18,temp.startpos_i);
end

% Make the shift
temp1 = INSPVAData;
clear INSPVAData;
INSPVAData =
[temp1(:,temp.startpos_i:length(temp1(2,:))),temp1(:,1:temp.startpos_i-
1)];
clear temp1
temp1 = COVData;
clear COVData;
COVData    =
[temp1(:,temp.startpos_i:length(temp1(2,:))),temp1(:,1:temp.startpos_i-
1)];
clear temp1

temp.mapstart_i  = floor(temp.mapstart*length(INSPVAData(2,:)));
if(temp.mapstart_i == 0); temp.mapstart_i = 1; end
temp.mapend_i    = floor(temp.mapend*length(INSPVAData(2,:)));
startx = INSPVAData(15,1);
starty = INSPVAData(16,1);
startz = INSPVAData(17,1);

% Discretize to the desired par.dx
map.D     = INSPVAData(18,temp.mapstart_i:temp.mapend_i) -
              INSPVAData(18,temp.mapstart_i);
D1d = [0:par.dx:max(map.D)];
```

```
INSPVAData =
Script_Adams_interp1(map.D',INSPVAData(:,temp.mapstart_i:temp.mapend_i)',D
   1d')';
COVData =
Script_Adams_interp1(map.D',COVData(:,temp.mapstart_i:temp.mapend_i)',D1d'
)';
clear temp1 D1d

% Filter the data
map.dx = par.dx;
if(par.ffilt*map.dx < 1)
    temp.inspva = INSPVAData; clear INSPVAData;
    [b,a] = butter(2,par.ffilt*map.dx,'low');
    INSPVAData(1:5,:) = temp.inspva(1:5,:);
    for j=6:11
        INSPVAData(j,:) = filtfilt(b,a,temp.inspva(j,:));
    end
    INSPVAData(12:size(temp.inspva,1),:) = temp.inspva(12:size(temp.inspva
,1),:);
    clear temp.inspva j a b;
end

% Seperate the data into meaningful variables
map.time  = INSPVAData(2,:);
map.lat   = INSPVAData(3,:);
map.long  = INSPVAData(4,:);
map.elev  = INSPVAData(5,:);
map.nvel  = INSPVAData(6,:);
map.evel  = INSPVAData(7,:);
map.uvel  = INSPVAData(8,:);
map.roll  = INSPVAData(9,:);
map.pitch = INSPVAData(10,:);
map.yaw   = INSPVAData(11,:);
map.odom  = INSPVAData(12,:) - INSPVAData(12,1);
map.x     = INSPVAData(15,:) - startx;
map.y     = INSPVAData(16,:) - starty;
map.z     = INSPVAData(17,:) - startz;
% map.D     = map.odom;
map.D     = INSPVAData(18,:) - INSPVAData(18,1);
map.covx  = COVData(4,:);
map.covy  = COVData(7,:);
map.covz  = COVData(11,:);
clear INSPVAData COVData


%-------------------------------------------------------------------------
% Load frag data
load('452ParkLn2_frag.mat');

% Shift data to starting position
temp.startpos_i  = floor(temp.startpos*length(INSPVAData(2,:)));
if(temp.startpos_i == 0);
    temp.startpos_i = 1;
    temp.added_odom = 0;
else
temp.added_odom = sqrt((INSPVAData(15,temp.startpos_i)-
```

```
    (INSPVAData(15,temp.startpos_i-1)))^2 +...
                        (INSPVAData(16,temp.startpos_i)-
    (INSPVAData(16,temp.startpos_i-1)))^2 +...
                        (INSPVAData(17,temp.startpos_i)-
    (INSPVAData(17,temp.startpos_i-1)))^2);

% Correct the odom and D before shifting
INSPVAData(12,1:temp.startpos_i-1) = INSPVAData(12,1:temp.startpos_i-1) +
     INSPVAData(12,length(INSPVAData(2,:))) + temp.added_odom;
INSPVAData(12,:) = INSPVAData(12,:) - INSPVAData(12,temp.startpos_i);
INSPVAData(18,1:temp.startpos_i-1) = INSPVAData(18,1:temp.startpos_i-1) +
     INSPVAData(18,length(INSPVAData(2,:))) + temp.added_odom;
INSPVAData(18,:) = INSPVAData(18,:) - INSPVAData(18,temp.startpos_i);
end

% Make the shift
temp1 = INSPVAData;
clear INSPVAData;
INSPVAData =
[temp1(:,temp.startpos_i:length(temp1(2,:))),temp1(:,1:temp.startpos_i-
1)];
clear temp1
temp1 = COVData;
clear COVData;
COVData    =
[temp1(:,temp.startpos_i:length(temp1(2,:))),temp1(:,1:temp.startpos_i-
1)];
clear temp1

 % Clip the frag to the desired percentages
temp.fragstart_i  = floor(temp.fragstart*length(INSPVAData(2,:)));
if(temp.fragstart_i == 0); temp.fragstart_i = 1; end
temp.fragend_i    = floor(temp.fragend*length(INSPVAData(2,:)));
temp1 = INSPVAData; clear INSPVAData; INSPVAData =
temp1(:,temp.fragstart_i:temp.fragend_i); clear temp1
temp1 = COVData; clear COVData; COVData =
temp1(:,temp.fragstart_i:temp.fragend_i); clear temp1

 % Filter the data
frag.dx = mean(diff(INSPVAData(18,:)));
if(par.ffilt*frag.dx < 1)
    temp.inspva = INSPVAData; clear INSPVAData;
    [b,a] = butter(2,par.ffilt*frag.dx,'low');
    INSPVAData(1:5,:) = temp.inspva(1:5,:);
    for j=6:11
        INSPVAData(j,:) = filtfilt(b,a,temp.inspva(j,:));
    end
    INSPVAData(12:size(temp.inspva,1),:) = temp.inspva(12:size(temp.inspva
,1),:);
    clear temp.inspva j a b;
end


% Seperate the data into meaningful variables
frag.time  = INSPVAData(2,:);
frag.lat   = INSPVAData(3,:);
```

```
frag.long  = INSPVAData(4,:);
frag.elev  = INSPVAData(5,:);
frag.nvel  = INSPVAData(6,:);
frag.evel  = INSPVAData(7,:);
frag.uvel  = INSPVAData(8,:);
frag.roll  = INSPVAData(9,:);
frag.pitch = INSPVAData(10,:);
frag.yaw   = INSPVAData(11,:);
frag.odom  = INSPVAData(12,:) - INSPVAData(12,1);
frag.x     = INSPVAData(15,:) - startx;
frag.y     = INSPVAData(16,:) - starty;
frag.z     = INSPVAData(17,:) - startz;
frag.D     = frag.odom;
frag.covx  = COVData(4,:);
frag.covy  = COVData(7,:);
frag.covz  = COVData(11,:);
clear INSPVAData COVData fragstart fragend startx starty startz


%-------------------------------------------------------------------------
% Find the offest in D for the frag data
temp.junk = sqrt((map.x - frag.x(1)).^2+(map.y - frag.y(1)).^2);
[junk,number] = min(temp.junk(1:floor(length(temp.junk)/2)));
if(number==1)
    temp.junkx = linspace(map.x(1),map.x(number+1),200);
    temp.junky = linspace(map.y(1),map.y(number+1),200);
elseif(number==length(map.x))
    temp.junkx = linspace(map.x(number-1),map.x(end),200);
    temp.junky = linspace(map.y(number-1),map.y(end),200);
else
    temp.junkx = linspace(map.x(number-1),map.x(number+1),200);
    temp.junky = linspace(map.y(number-1),map.y(number+1),200);
end
temp.junk_2 = sqrt((temp.junkx - frag.x(1)).^2+(temp.junky -
frag.y(1)).^2);
[junk,number_2] = min(temp.junk_2);
frag.offset = map.D(number);

%-------------------------------------------------------------------------
% Create measurements incremented every 'par.travel' meters
if(par.travel*par.N > frag.odom(end))
    par.N = floor(frag.odom(end)/par.travel) - 2;
end
locs = [];
for i=1:par.N-1
    [junk,locs(i)] = find(frag.odom - i*par.travel >= 0,1);
end
locs = [1,locs];
frag.x_m     = frag.x(locs);
frag.y_m     = frag.y(locs);
frag.z_m     = frag.z(locs);
frag.time_m  = frag.time(locs);
frag.lat_m   = frag.lat(locs);
frag.long_m  = frag.long(locs);
frag.elev_m  = frag.elev(locs);
frag.nvel_m  = frag.nvel(locs);
```

```
frag.evel_m  = frag.evel(locs);
frag.uvel_m  = frag.uvel(locs);
frag.roll_m  = frag.roll(locs);
frag.pitch_m = frag.pitch(locs);
frag.yaw_m   = frag.yaw(locs);
frag.odom_m  = frag.odom(locs);
frag.odom_m  = [0,frag.odom_m(2:end) - frag.odom_m(1:end-1)];
frag.D_m     = frag.D(locs);
frag.covx_m  = frag.covx(locs);
frag.covy_m  = frag.covy(locs);
frag.covz_m  = frag.covz(locs);
clear i junk locs
%%-------------------------------------------------------------------------
```

Appendix 3: MATLAB code for evaluating particle parameters at each iteration

```
%-------------------------------------------------------------------------
% If the particles are moved off the map then loop them around the track
% (or the beginning of the map)
temp.maxD                 = max(map.D);
temp.minD                 = min(map.D);
[junk,indecies2]          = find(xu.long(ind,:) > min(temp.maxD));
xu.long(ind,indecies2)    = xu.long(ind,indecies2) - (min(temp.maxD)-
                            max(temp.minD));
[junk,indecies3]          = find(xu.long(ind,:) < max(temp.minD));
xu.long(ind,indecies3)    = xu.long(ind,indecies3) + (min(temp.maxD)-
                            max(temp.minD));
clear junk indecies2 indecies3

%-------------------------------------------------------------------------
% Calculate particle parameters
temp.interps2 =
Script_Adams_interp1(map.D',[map.pitch',map.roll',map.x',map.y'],xu.long(i
nd,1:par.S)')';
particle.x = temp.interps2(3,:);
particle.y = temp.interps2(4,:);

%-------------------------------------------------------------------------
% Calculate the estimated position and parameters
prediction.long(index)       = mean(xu.long(ind,:),2);
temp.interps                 =
Script_Adams_interp1(map.D',[map.pitch',map.roll',map.x',map.y'],predictio
n.long(index));
prediction.pitch(index)      = temp.interps(1);
prediction.x(index)          = temp.interps(3);
prediction.y(index)          = temp.interps(4);
prediction.error_mag(index)  = sqrt((prediction.x(index)-
                               frag.x_m(index)).^2+(prediction.y(index)-
                               frag.y_m(index)).^2);
prediction.error_long(index) = abs(prediction.long(index) - frag.D(index)
                               - frag.offset);

%-------------------------------------------------------------------------
% Calculate and correct for the lane error
temp.lane_errors = sqrt((map.x - frag.x_m(index)).^2+(map.y -
                   frag.y_m(index)).^2);
[junk,number] = min(temp.lane_errors);
if(number==1)
    temp.X_norm = linspace(map.x(1),map.x(number+1),200);
    temp.Y_norm = linspace(map.y(1),map.y(number+1),200);
elseif(number==length(map.x))
    temp.X_norm = linspace(map.x(number-1),map.x(end),200);
    temp.Y_norm = linspace(map.y(number-1),map.y(end),200);
else
    temp.X_norm = linspace(map.x(number-1),map.x(number+1),200);
    temp.Y_norm = linspace(map.y(number-1),map.y(number+1),200);
```

```
end
temp.lane_errors_2 = sqrt((temp.X_norm - frag.x_m(index)).^2+(temp.Y_norm
                    - frag.y_m(index)).^2);
[temp.lane_error(index),number_2] = min(temp.lane_errors_2);
frag.x_m2map(index) = temp.X_norm(number_2);
frag.y_m2map(index) = temp.Y_norm(number_2);
prediction.error_actual(index) = sqrt((prediction.x(index)-
                        frag.x_m2map(index)).^2+(prediction.y(index)-
frag.y_m2map(index)).^2);
prediction.error_lane_keeping(index) = sqrt((frag.x_m(index)-
                        frag.x_m2map(index)).^2+(frag.y_m(index)-
frag.y_m2map(index)).^2);
clear junk number number_2;
```

Appendix 4: MATLAB code for the re-sampling step

```
temp.particle_long = xu.long(ind,:);
C = cumsum(particle.q(v,:));
u(1) = rand(1)/par.S;
u(2:par.S) = u(1) + cumsum(ones(1,par.S-1)/par.S);
indecies(1:par.S) = 0;
i = 1;
for j=1:par.S
    while(u(j)>C(i))
        i = i+1;
    end
    indecies(j) = i;
end
particle.long(v,:) = temp.particle_long(indecies);
clear C u j i indecies
```

Appendix 5: MATLAB code for plots

```
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
% Plot the progression of the particles:
plots.width = 5;
plots.height = 6;
plots.fontsize = 12;
plots.subwidth = 0.335;
plots.subheight = 0.2792;
plots.posx = [0.13 0.57 0.13 0.57];
plots.posy = [0.52 0.52 0.1283 0.1283];
plots.num_index =
              [1,floor(0.15*par.N),floor(0.3*par.N),floor(0.45*par.N)];
plots.legend_loc = [0.5,1-(1-
                           (plots.posy(1)+plots.subheight))/2,0.1,0.0169];

temp.div=1000;

%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
% Longitudinal Error plot
figure('Color',[1 1 1],'Units','inches','Position',[1, 3, 6, 5])
h1 = semilogy(frag.D_m,prediction.error_long,'b','LineWidth',2);
hold on;
h2 = semilogy([0,frag.D_m(par.N)],[par.dx,par.dx],'k-.','LineWidth',2);
set(gca,'XLim',[0,frag.D_m(par.N)])
ylabel('Position Estimate Error (m)','FontSize',12)
xlabel('Distance Traveled (m)','FontSize',12)
hold off
set(gca,'FontSize',12)
clear h1 h2 i
```

REFERENCES

[1] **M. Heron, D.L. Hoyert, S.L. Murphy, J. Xu, K.D. Kochanek, and B. Tejada-Vera**, "Deaths: Final data for 2006," *National Vital Statistics Reports*, vol.57, no.14.

[2] **J.C. Gerdes and E.J. Rossetter**, "A Unified Approach to Driver Assistance Systems Based on Artificial Potential Fields." ASME *Journal of Dynamic Systems, Measurement, and Control*, 123:431-438, September 2001.

[3] **D.M. Bevley, J.C. Gerdes, C. Wilson, and G. Zhang**, "The use of GPS-based velocity measurements for improved vehicle state estimation", *Proc. American Control Conf.*, pp.2538-2542, 2000.

[4] **Z. Sun, G. Bebis, and R. Miller**, "On-road vehicle detection: a review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 5, pp. 694-711, 2006.

[5] (GPSwebsite)

[6] **I. Skog**, "A low-cost GPS Aided Inertial Navigation System for Vehicular Applications", Master's thesis, Royal Institute of Technology, Sweden, Mar. 2005, IR-SB-EX-0506.

[7] **A. Pinker and C. Smith**, "Vulnerability of the GPS Signal to Jamming", GPS Solutions, Vo. 3, No.2, 1999.

[8] **M. E. E. Najjar and P. Bonnifait**, "A road-matching method for precise vehicle localization using belief theory and kalman filtering," *Auton. Robots*, vol. 19, no. 2, pp. 173-191, 2005.

[9] **T. Oskiper, Z. Zhu, S. Samarasekera, and R. Kumar**, "Visual odometry system using multiple stereo cameras and inertial measurement unit," in Computer Vision and Pattern Recognition, 2007. CVPR '07. *IEEE Conference on*, Minneapolis, MN, USA, Jun. 17-22, 2007, pp. 1-8.

[10] **M. Bosse and R. Zlot**, "Keypoint design and evaluation for place recognition in 2D lidar maps." Zurich, Switzerland: s.n., 2008.

[11] **H.-C. Chu and R.-H. Jan**, "A GPS-less self-positioning method for sensor networks," 2005. Proceedings. *11th International Conference on Parallel and Distributed Systems*, vol. 2, pp. 629-633, Jul. 20-22, 2005.

[12] **A. Dean, R. Martini, and S. Brennan**. "Terrain-Based Road Vehicle Localization Using Particle Filters." *Proceedings of the 2008 American Control Conference*, Seattle, WA, Jun. 11-13, 2008, pg 236-241.

[13] **F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. J. Nordlund**, "Particle filters for positioning, navigation, and tracking," *Signal Processing IEEE Transactions on[see also Acoustics, Speech, and Signal Processing, IEEE Transactions on*], vol. 50, no. 2, pp. 425-437, Feb. 2002.

[14] **A. Dean**,"Terrain-based Road Vehicle Localization using Attitude Measurements", Ph.D. Thesis, Mechanical Engineering, The Pennsylvania State University, October 2008.

[15] **A. Dean, P. Vemulapalli, and S. Brennan**. "Highway Evaluation of Terrain-Aided Localization Using Particle Filters." Proceedings of the 2008 *ASME Dynamic Systems and Control Conference*, Ann Arbor, MI, Oct. 20-22, 2008.

[16] **A. Dean, J. Langelaan, and S. Brennan**. "Initializing An Unscented Kalman Filter Using A Particle Filter." *Proceedings of the 2009 Dynamic Systems and Control Conference*, Hollywood, CA, USA, October 12-14, 2009.

[17] **J. Haitsma, T. Kalker**, "A highly robust audio fingerprinting system." 2002: Proceedings of *International Conference on Music Information Retrieval*.

[18] **S. Baluja, M. Covell**, "Waveprint: Efficient Wavelet-Based Audio Fingerprinting." 2008, Issue Pattern Recognition.

[19] **Y. Ke, D. Hoiem, R. Sukthankar**, " Computer vision for music identification", s.l. : *Proceedings of Computer Vision and Pattern Recognition*, 2005.

[20] **C. Burges, J. Platt, S. Jana**, "Distortion discriminant analysis for audiofingerprinting." s.l. : IEEE *Trans. Speech & Audio Processing*, 2003.

[21] **C. Burges, J. Platt, S. Jana**, "Extracting noise-robust features from audio data." s.l. : ICASSP, 2002.

[22] **P. Vemulapalli, A. Dean, and S. Brennan**, "Pitch-based Vehicle Localization using Time Series Subsequence Matching with Multi-scale Extrema Features", To appear in *Proceedings of the 2011 American Control Conference*. June 29-July 01, San Francisco.

[23] **K. Li, Han-Shue Tan, and J. Karl Hedrick**., "Map-Aided GPS/INS Localization Using a Low-Order Constrained Unscented Kalman Filter." Shanghai : *Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, 2009.

[24] **A. Dean, J. Langelaan, and S. Brennan**. "Initializing An Unscented Kalman Filter Using A Particle Filter." *Proceedings of the 2009 Dynamic Systems and Control Conference*, Hollywood, CA, USA, October 12-14, 2009.

[25] **A. Dean, P. Vemulapalli, and S. Brennan**. "Highway Evaluation of Terrain-Aided Localization Using Particle Filters." Proceedings of the 2008 *ASME Dynamic Systems and Control Conference*, Ann Arbor, MI, Oct. 20-22, 2008.

[26] **J. K. Uhlmann**, "Algorithms for multiple-target tracking." *American Scientist, Comm. of the ACM*, 18(9), 1975.

[27] **K. Mikolajczyk, H. Uemura**, "Action recognition with motion-appearance vocabulary forest," *CVPR*, 2008.

[28] **S. Thrun, D. Fox, and W. Burgard**, " A probabilistic approach to concurrent mapping and localization for mobile robots", *Machine Learning*, 31, 1998.

[29] **Y. Liu and S. Thrun**, "Results for outdoor-SLAM using sparse extended information filters," in *Proc. IEEE Int. Conf. Robot. Autom*., Taipei, Taiwan, R.O.C., 2003, pp. 1227-1233.

[30] **S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A.Y. Ng**, "Simultaneous mapping and localization with sparse extended information filters," *Proc. WAFR*, 2002.

[31] **M. Bosse and R. Zlot**, "Keypoint design and evaluation for place recognition in 2D lidar maps." Zurich, Switzerland: s.n., 2008.

[32] **G. Schindler, M. Brown and R. Szeliski**, "City-scale location recognition." Anchorage, Alaska : *IEEE* Computer Society Conference on Computer Vision and Pattern Recognition, 2007.

[33] **D. Fontanelli, L. Ricciato, and S. Soatto**, "A fast ransac-based registration algorithm for accurate localization in unknown environments using lidar measurements," in 2007 *IEEE Int. Conf. on Automation Science and Engineering*, 2007, pp. 597-602.

[34] **R. Martini**, "GPS/INS Sensing Coordination for Vehicle State Identification and Road Grade Positioning," M.S. Thesis, Mechanical and Nuclear Engineering, The Pennsylvania State University, Aug 2006.

[35] **J. P. Golden**, "Terrain contour matching/TERCOM/- A cruise missile guidance aid," Image processing for missile guidance, pp. 10-18, 1980.

[36] **S. Williams, G. Dissanayake, and H. Durrant-Whyte**, "Towards terrain-aided navigation for underwater robotics," Advanced Robotics, vol. 15, no. 5, pp. 533-549, 2001.

[37] **L. Ledwich and S. Williams**,"Reduced SIFT Features For Image Retrieval and Indoor Localisation," Australasian Conf. on Robotics and Automation, Can berra, 2004.

[38] **K. Murphy, A. Torralba, D. Eaton and W.T. Freeman**: Object Detection and Localization Using local and Global Features. Sicily Workshop on Object Recognition, Lecture Notes in Computer Science, (2005), 393-413.

[39] **R. Hockney**, 1996. "The Science of Computer Benchmarking", *Society for Industrial and Applied Mathematics*,3600 University City Science Center, Philadelphia, PA 19104-2688, Chap. 2, pp. 21-22.

[40] **S. Baluja, M. Covell**, "Waveprint: Efficient Wavelet-Based Audio Fingerprinting," 2008, Issue Pattern Recognition.

[41] **Y. Ke, D. Hoiem, R. Sukthankar**, " Computer vision for music identification," s.l. : Proceedings of Computer Vision and Pattern Recognition, 2005.

[42] **N. Ozay, M. Sznaier, C. Lagoa and O. Camps**, "A Sparsification Approach to Set Membership Identification of a Class of Affine Hybrid Systems," Proceedings of *IEEE Conference on Decision and Control*, 2008.