

**The Utilization of Geometric Hashing Techniques for Feature Association during
Ground Vehicle Localization**

by

Michael Sprunk

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 5, 2023

Keywords: geometric hashing, feature-based localization, feature-based maps, map integrity,
data association

Copyright 2023 by Michael Sprunk

Approved by

David M. Bevly, Chair, McNair Endowed Distinguished Professor of Mechanical Engineering
Scott Martin, Assistant Research Professor of Mechanical Engineering
Luke Oeding, Associate Professor of Mathematics and Statistics

Abstract

This thesis presents a new approach to laser-based robot localization using the concept of geometric hashing. The technique operates on an *a priori* feature map representing the navigation environment by recording all possible feature combinations as transformation-invariant sets. Each combination can be defined by an implementation-specific geometric basis, and then hashed in order to promote rapid parallelizable search conditions and easily encode large quantities of information.

The primary focus of this work is centered on the individual component of map data association within the much larger localization pipeline. At this step, the navigation system is generally required to provide a correct association between features extracted in real-time from the environment and their corresponding *a priori* mapped counterparts. In addition to the main concerns of accuracy and reliability, this thesis addresses several other relevant challenges present in feature-based localization such as time complexity, sensitivity to noise, and the detection of map symmetries.

The concept of using geometric hashing for laser-based localization consists of three phases: the training phase, screening phase, and recognition phase. Within this work, each phase is thoroughly defined and analyzed. Particular attention is given toward the utilization of cylindrical-like features found predominantly in urban environments for use during localization. A simulation was developed to test and verify geometric hashing localization in both unique and ambiguous environments. The results validated that geometric hashing localization can provide sub-meter level accuracy even in the presence of ambiguous geometries so long as sufficient information is present. To test the approach on time-critical scenarios, an implementation of the data association algorithm written C++ was integrated into an existing localization framework deployed on a vehicle. Results showed that the positioning solution is capable of providing sub-meter accuracy at 20 Hz update rates driving through urban environments.

Acknowledgements

Diese Diplomarbeit wäre ohne das RD/ASF der Daimler AG nicht möglich gewesen. Mein herzlichster Dank gilt dem Team der Fahrzeugpositionierung für all ihre Hilfe und Unterstützung. Ich bin dankbar, dass ich während meiner Zeit in Stuttgart die Gelegenheit hatte, zusammenzuarbeiten und so viel zu lernen.

In particular, I would like to thank my supervisors Isabell Hofstetter and Florian Ries. It was a pleasure working together on this topic and I really appreciate all the guidance, patience, and encouragement. In addition, I would like to thank several members of the MRT group from the Karlsruhe Institute of Technology for providing this work with the prior software framework, generous amounts of data to work with, and taking the time to answer questions when needed.

I'd like to acknowledge my committee members: Dr. David Bevly, Dr. Scott Martin, Dr. George Flowers, Dr. Luke Oeding, and Dr. John Hung (who has retired before I could finish writing). Each of you have played an important role in my academic growth at Auburn. I am grateful to have studied under each of you and thank you for supporting me during my time in the GAVLAB. I would personally like to thank my committee chair, Dr. Bevly, for allowing me the opportunity to not only study in the GAVLAB at Auburn, but also intern with Daimler AG. Dr. Bevly has had the most profound impact on my current career direction and passion for robotics, and I am eternally grateful for all the experiences I have gained studying under his leadership.

I'd also like to thank all the members of the GAVLAB who I have interacted with over the years. I am so grateful to have met and worked with so many of you and I am thankful for all the bonds we share through late nights and hard work. In particular, I would like to thank Ryan Shaw and Ethan Edwards for their relentless encouragement and support to finish this effort. Without either of them, I would not be the person I am today.

Lastly, I would like to thank my parents, Joanne and Erwin Sprunk. Both of you have played a foundational role in my interest in engineering and desire to study at the highest level of education. Thank you both for giving me the opportunity to exercise my passion and constantly equipping me with the tools I need to achieve my goals. I love you both very much.

Table of Contents

Abstract	ii
Acknowledgements	iii
List of Figures	ix
List of Tables	xiii
List of Keywords	xvi
1 Introduction	1
1.1 Background on Feature-based Localization	2
1.1.1 Model-based Recognition	3
1.2 Prior Work	6
1.3 Contributions	7
1.4 Outline	8
2 Geometric Hashing	10
2.1 Basis Formulation	10
2.2 Hashing Functions	14
2.3 Invariant Matching	18
2.4 Other Characteristics of Geometric Hashing	23
2.4.1 Concurrent Processing	23
2.4.2 Sensitivity to Noise	24

2.4.3	Non-uniform Hash Distribution	26
2.4.4	Detection of Feature Symmetries	27
3	Detecting Ambiguities in Feature Maps	28
3.1	Feature Extraction	29
3.1.1	Map Generation	31
3.2	Map Symmetries	31
3.2.1	Detecting Similarities	34
3.2.2	Forming Constellations	37
3.3	Insight on Map Integrity	38
4	Geometric Hashing in Localization	40
4.1	Offline Processing	40
4.1.1	The Training Phase	41
4.1.1.1	Basis Parameters	41
4.1.1.2	Quantization and Hashing	43
4.1.1.3	Collision Filtering	44
4.1.1.4	Algorithm: Training	46
4.1.2	The Screening Phase	47
4.1.2.1	Congruent Bases	47
4.1.2.2	Algorithm: Screening	48
4.2	Localization Pipeline	51
4.2.1	The Recognition Phase	51
4.2.1.1	Noise Mitigation	53
4.2.2	Candidate Selection Methods	54
4.2.2.1	Algorithm: Recognition	60
4.2.3	Verifying the Association	60

5	Simulation and Experimental Test Design	63
5.1	Hardware Setup	63
5.2	The Reference Solution	66
5.3	Mapped Circuits	67
5.3.1	Sindelfingen Route	67
5.3.2	Karlsruhe Route	68
5.4	Simulation Framework	73
5.5	Localization Framework	75
6	Experimental Results	77
6.1	Simulation	77
6.1.1	S-bend East – No Ambiguities	79
6.1.2	Ambiguity Analysis – Longer S-bend East	85
6.1.3	Discussion of Results for S-bend East	99
6.2	Integrated Localization Solution	101
6.2.1	Sindelfingen	103
6.2.1.1	Fine Resolution – Sifi5	103
6.2.1.2	Coarse Resolution – Sifi20P	112
6.2.1.3	Discussion of Sindelfingen Results	121
6.2.2	Karlsruhe	127
6.2.2.1	Fine Resolution – KIT5	128
6.2.2.2	Course Resolution – KIT20	138
6.2.2.3	Discussion of Karlsruhe Results	147
7	Conclusion and Future Work	153
7.1	Conclusions on Geometric Hashing	153
7.2	Future Work	155

7.2.1	Additional Primitives and Descriptors	155
7.2.2	Integrity Risk Monitor	156
	References	157
	Appendices	161
	Appendix A	163

List of Figures

2.1	Model of extracted point features	12
2.2	Creation of Invariants through Rigid Transformation	13
2.3	Quantization of invariants with $q_{pose} = 1$	16
2.4	”Stacking” of bases to form layers	19
2.5	Scene of local environment containing extracted point features	20
2.6	Rigid transformation of local scene	21
2.7	Recognition of model within the scene	22
3.1	Sample feature map with symmetry	32
3.2	Highlighting a symmetric polygon in feature map	33
3.3	Layer comparison in the basis domain for bucket b	35
4.1	Hash Collision Resolution Methods - Bin Size: 0.5m	45
4.2	Model example depicting congruent base scenario	48
4.3	Congruent bases formed via two overlapping basis origin points	49
4.4	Geometric hashing localization flowloop	52
5.1	BerthaOne robotic vehicle depicting computing hardware configuration [1].	64
5.2	BerthaOne sensory field of view and layout [1].	65
5.3	Driven route in Sindelfingen, Germany	68
5.4	Extracted cylinders from Sindelfingen route	69
5.5	Derived speed of vehicle with smoothing	70
5.6	Driven route in Karlsruhe, Germany	70
5.7	Extracted cylinders from Karlsruhe route	71

5.8	Derived speed of vehicle with smoothing	72
6.1	SENA: Overlay of odometry with map of landmarks	80
6.2	SENA: Planar position, heading, and velocity of the vehicle over time	81
6.3	SENA: Error between reference solution and odometry	82
6.4	SENA: Missed and Incorrect associations at each observation event	83
6.5	SENA: Detections, Associations, and Verifications at each observation event	84
6.6	SELA: Overlay of odometry with map of landmarks	86
6.7	SELA: Planar position, heading, and velocity of the vehicle over time	87
6.8	SELA: Error between reference solution and odometry	88
6.9	SELA: Missed and Incorrect associations at each observation event	89
6.10	SELA: Detections, Associations, and Verifications at each observation event	90
6.11	SELA: Snapshot of an observation frame where an incorrect association occurred	91
6.12	SELA: Heatmap representing ambiguous constellation density (by centroid) over mapped area	92
6.13	SELA: View all ambiguous constellations of 6 vertices	93
6.14	SELA: View all ambiguous constellations of 3 vertices	94
6.15	SELA: Heatmap representing translational error from ambiguous features over mapped area	95
6.16	SELA: Heatmap representing rotational error from ambiguous features over mapped area	96
6.17	SELA: Probability of correct association given ambiguous features	97
6.18	SELA: Probability of correct association given layer candidates	98
6.19	SIFI5: Overlay of odometry with map of landmarks	104
6.20	SIFI5: Breakdown of X position with update availability	105
6.21	SIFI5: Breakdown of Y position with update availability	106
6.22	SIFI5: Breakdown of heading with update availability	107
6.23	SIFI5: Planar position and heading of the vehicle over time	108

6.24	SIFI5: Error between reference solution and odometry	109
6.25	SIFI5: Measurement correction availability during the drive	110
6.26	SIFI5: Processing time for each detection during the drive	111
6.27	SIFI20: Overlay of odometry with map of landmarks	113
6.28	SIFI20: Breakdown of X position with update availability	114
6.29	SIFI20: Breakdown of Y position with update availability	115
6.30	SIFI20: Breakdown of heading with update availability	116
6.31	SIFI20: Planar position and heading of the vehicle over time	117
6.32	SIFI20: Error between reference solution and odometry	118
6.33	SIFI20: Measurement correction availability during the drive	119
6.34	SIFI20: Processing time for each detection during the drive	120
6.35	SIFI5: Heatmap representing ambiguous constellation density (by centroid) over mapped area	122
6.36	SIFI5: View all ambiguous constellations of 3 vertices	123
6.37	SIFI5: Heatmap representing translational error from ambiguous features over mapped area	124
6.38	SIFI5: Heatmap representing rotational error from ambiguous features over mapped area	125
6.39	Incorrect feature association caused by ambiguous geometry	128
6.40	KIT5: Overlay of odometry with map of landmarks	130
6.41	KIT5: Breakdown of X position with update availability	131
6.42	KIT5: Breakdown of Y position with update availability	132
6.43	KIT5: Breakdown of heading with update availability	133
6.44	KIT5: Planar position and heading of the vehicle over time	134
6.45	KIT5: Error between reference solution and odometry	135
6.46	KIT5: Measurement correction availability during the drive	136
6.47	KIT5: Processing time for each detection during the drive	137

6.48	KIT20: Overlay of odometry with map of landmarks	139
6.49	KIT20: Breakdown of X position with update availability	140
6.50	KIT20: Breakdown of Y position with update availability	141
6.51	KIT20: Breakdown of heading with update availability	142
6.52	KIT20: Planar position and heading of the vehicle over time	143
6.53	KIT20: Error between reference solution and odometry	144
6.54	KIT20: Measurement correction availability during the drive	145
6.55	KIT20: Processing time for each detection during the drive	146
6.56	KIT5: Heatmap representing ambiguous constellation density (by centroid) over mapped area	149
6.57	KIT5: View all ambiguous constellations of 3 vertices	150
6.58	KIT5: Heatmap representing translational error from ambiguous features over mapped area	151
6.59	KIT5: Heatmap representing rotational error from ambiguous features over mapped area	152
A.1	Example structure of Hash Table	164
A.2	Example structure of Layers Database	165

List of Tables

2.1	Structure of hash entries	17
3.1	Generated Similarity Sets for layers 2 and 4	36
3.2	Ambiguous Constellations for bucket b	39
6.1	Simulated Vehicle Parameters	78
6.2	Simulated LiDAR Parameters	78
6.3	Kalman Filter Parameters	78
6.4	SENA: Parameters	79
6.5	SELA: Parameters	85
6.6	SENA: Error metrics	99
6.7	SELA: Error metrics	99
6.8	SENA: Result statistics	99
6.9	SELA: Result statistics	100
6.10	SIFI5: Parameters	103
6.11	SIFI20: Parameters	112
6.12	SIFI5: Error metrics	121
6.13	SIFI20: Error metrics	121
6.14	SIFI5: Result statistics	126
6.15	SIFI20: Result statistics	127
6.16	KIT5: Parameters	129
6.17	KIT20: Parameters	138
6.18	KIT5: Error metrics	147

6.19 KIT20: Error metrics	147
6.20 KIT5: Result statistics	148
6.21 KIT20: Result statistics	149

List of Algorithms

4.1	Training Phase	46
4.2	Screening Phase	50
4.3	Associate Features 1	55
4.4	Associate Features 2	56
4.5	Associate Features 3	57
4.6	Associate Features 4	58
4.7	Associate Features 5	59
4.8	Associate Features 6	59
4.9	Recognition Phase	61

List of Keywords

Similarity Set a set of invariant features that is geometrically congruent across multiple bases

Ambiguous Constellation a unique similarity set within a bucket

Primitive a feature defined by fundamental geometries such as points, edges, and curves

Model a collection of semantic features constituting an object or image desired to be recognized

Hash Function an algorithm that maps arbitrarily-sized data to a fixed-size

Hash Code a uniquely deterministic fixed-sized output from a hash function

Geometric Basis a coordinate frame defined by primitive features

Invariant a feature that remains geometrically congruent after applying a transformation

Bucket a data structure containing information relating to a hash code

Hash Entry a key-value pair consisting of a hash code and a bucket

Hash Table a data structure consisting of hash entries

Basis Congruency A phenomenon where the quantized basis origin of two layers overlap along with all quantized invariants

Bin a discrete quantity containing continuous data

Feature Map a collection arbitrary features and their positions relative to an origin

Hash Collision a phenomenon in which two independent features hash to the same value

Layer an individual geometric basis within a collection of bases

Neighborhood a proximity group of hash entries within the hash table

Probe the creation of a geometric basis for model matching during recognition

Chapter 1

Introduction

The rapid growth and development of technology in our society today has led to an increased inclusion of autonomous systems into a wide variety of applications. As such, with each new application there exists a fresh set of challenges that must be considered and understood in order to develop safe, reliable, and efficient solutions. In many cases, the investigation of these challenges drive research efforts to continue the cycle and further push the limits of cutting-edge technology. Other scenarios express a need for the creation of original solutions to meet the often unique demands of application-specific requirements. One of the most recurring challenges spanning over many different autonomous applications is the task of localization, or in other words, the autonomous system understanding it's own position and orientation.

Although there are many localization approaches that exist in both research and industry today, this thesis addresses the numerous challenges associated with feature-based localization. In particular, the thesis will discuss the approach and implementation of a model-based technique called geometric hashing for use in the real-time localization of ground vehicles. This approach of geometric hashing requires *a priori* information of the environment and seeks to provide an association of features gathered from a local scene to a corresponding location within the mapped environment. By obtaining a correct association, the precise position and orientation of the system is theoretically known, and localization can successfully be achieved.

However, previous research suggests that this task is often not trivial and has high potential to provide misleading information to the system [2]. Further, many applications are unable to detect the presence of misleading information until after it is wrongfully utilized. For safety-critical applications, the inclusion of misleading information is particularly hazardous and the consequences could lead to system damage, serious injury, or even death. With this information in mind, a need arises to compute, monitor, and incorporate the risk of misleading information before providing candidate associations.

1.1 Background on Feature-based Localization

Feature-based Localization is a specific flavor among various navigation techniques that utilize visual-based sensory information to define features, often called “landmarks”, from the surrounding environment. These features are selected such that they could be easily recognized at a later time and provide positional information to a system within the bounded environment. Typically, systems that employ feature-based localization approaches obtain visually dense information from cameras or LiDAR. However, prior research has also investigated the use of RADAR and SONAR for feature-based applications [3, 4].

As the title “feature-based” suggests, applications of these techniques are inherently dependent on prior information about the entities to be recognized. Each contribution of prior information gathered should provide, at a minimum, a distinguishable feature to be recognized as well as positional information indicating where the feature is relative to an arbitrary origin. The process of collecting information from an environment or region is typically referred to as *Map Generation*, and the data structure representing the aggregation of all informational pieces is known as a **feature map**. There are numerous techniques and challenges associated with generating maps and collecting high fidelity map features, but many of these are too complex to introduce here and are considered outside the scope of this work.

A well studied alternative to feature-based mapping approaches is using occupancy grid structures to aggregate a model of the world during online operation. In the research community, these approaches are typically used to solve the Simultaneous Localization and Mapping (SLAM) problem [5]. Several noteworthy differences from feature-based approaches are that SLAM typically does not require an *a priori* process in that both the map generation and localization are performed together online. However, this can introduce the problem of large memory requirements and low update rates due to the computational complexity of building the maps [6].

A typical scenario highlighting the framework of feature-based localization could be a car driving through an arbitrarily bounded urban region containing a series of intersections. Each intersection within the region provides a rich set of interesting features to capture such as signs,

street lights, and curbs. A feature map, or database of features, could be generated on an initial route through the region using a set of user-defined qualities of interest and positional metrics. On subsequent routes around the region, new features extracted via the same methods used to generate the map can be correlated with existing features in the database to provide a mapping between the vehicle's current field of view and the position of the vehicle within the mapped region.

In general, the Feature-based approach can be summarized into a four-step process: [7]

- Data Acquisition - receive a new batch of visual data
- Feature Extraction - define and extract features of interest
- Feature Association - correlate these features with prior features
- Verification - verify the correlation is correct

This work will focus primarily on the details and challenges of the feature association step of the procedure. However, some discussion of feature extraction and verification will also be provided throughout the thesis.

1.1.1 Model-based Recognition

Widening the scope beyond mobile platform localization, at the heart of the feature extraction and association steps lie the concept of model-based recognition. The use of model-based techniques to recognize simple objects has been gaining significant popularity for a wide variety of applications among the research community and autonomous system localization is not exempt from these. In general terms, recognition algorithms are considered to be model-based if they utilize previously obtained information, or **models**, to identify matching objects in a given scene. The definition of a model can vary among applications, but typical examples consist of one or a collection of semantic features that uniquely define an object or context to be recognized. Upon detecting these features, they can be decomposed into a collection of **primitive** geometries, such as points, edges, or curves, and then stored in a database [8, 9].

If executed properly for the given scenario, these model primitives can then be used to correspond with live sensory detections and relative transformation data can be obtained. However, this task is not always simple. Consequently, the majority of research suggests that there are four overarching challenges common to all variations of model-based recognition techniques:

- objects that have undergone various types of geometric transformations,
- objects that move or change frequently,
- objects that are partially occluded from view and,
- objects that are large in number or geometric complexity.

Each of these issues influence feasibility of using model-based techniques for reliable recognition requirements. Relating back to feature-based localization, each issue has a particular effect on the feature extraction and data association steps highlighted in Section 1.1.

Taking a look at each item individually, if an object in a scene is rotated or disfigured in a way that deviates significantly from the model, detection algorithms may be unable to extract the necessary features to provide the desired association. Further, even if the object was successfully detected, it still carries an increased likelihood to be mistaken with another feature, or remain unassociated on grounds that it is too different from the existing models. Similar phenomena can occur for moving and partially occluded objects as well, which necessitates the verification step following each association. Objects that occur too frequently, especially when in close proximity to one another, have a saturation effect during recognition. In other words, they introduce an ambiguity which all detection instances provide nearly equal likelihoods of a correct match with a single instance of a model. On the opposite side, objects that are geometrically complex have a tendency to be "too unique" and maybe be difficult to detect and recognize unless very specific criteria are met.

Additionally, as the number of models increases within the database, the ability to search for the correct match becomes an increasingly relevant concern [10]. To minimize complications from these challenges, a unique approach to searching for and compartmentalizing models is necessary. In order to solve the problem holistically, this approach would be tasked with accommodating all four challenges for any possible scenario. Fortunately, the geometric hashing algorithm provides strong foundation to accomplish this task.

1.2 Prior Work

During the early stages of computer vision research, the concepts of transformation-invariant features were first introduced as a recognition algorithm for 2D image objects that could be either partially occluded or under a geometric transformation [11]. Shortly after, the topic of object recognition turned towards utilizing model-based techniques. Along with the trend shift, transformation-invariant recognition was also applied to the model-based approach thus allowing for the recognition of 3D shapes appearing in 2D images. Amidst the thrust of research to find more reliable methods to detect and recognize 3D features from images, the concept of geometric hashing was introduced by *Lamdan et al* as a promising technique in the presence of noise and computational complexity [9, 8, 12]. The algorithm utilized a set of semantically chosen "interest points" to reliably recognize features that maintain affine congruency.

Since the initial concept, geometric hashing has been further studied and applied in several other works. A few notable examples from *Grimson et al* and *Lamdan et al* focus on the sensitivity of the recognition procedure when afflicted with noisy detections as well as the consequential error brought about by the noise [13, 14]. In his dissertation, *Rigoutsos* employs geometric hashing into a large parallel computational framework where features can undergo rigid, affine, or similarity transformations [7]. In his work, he also employs a Bayesian statistical approach in order to mitigate the affects noisy interest points present during recognition and obtain a quality metric to each association. *Tsai* utilizes both geometric hashing and the Hough Transform to extract line features from noisy images and continues to develop and apply Bayesian techniques to improve robustness in the presence of outliers [15].

Califano and *Mohan* compare index-based techniques like geometric hashing and the Hough Transform to discuss their shortcomings on highly correlated models and proposes using a higher dimensional approach to robustly recognize sub-components of models and utilize larger model databases [16]. *Gilbert* and *Bowden* construct a neural network based approach to marry the concept of geometric hashing with machine learning in attempt to reduce the combinatorial complexity of basis selection [17].

All of the prior art mentioned above focuses exclusively within the domain of recognizing features from imagery, yet there is not much research on using geometric hashing techniques with 3D LiDAR data. Further, none of the above present an application of the recognition approaches on a time and safety critical system. *Tomono* presents an effort to utilize geometric hashing techniques as a scan matching approach for mobile robot global localization and greatly benefits from avoiding a feature extraction algorithm since each laser return is used as a feature [18]. However, this approach is mainly suited for planar Laser Range Finders (LRF) and will become too computationally expensive working with the data throughput of modern 3D LiDAR sensors. Lastly, in his work, *Joerger* presents a thorough investigation of how to apply the model-based feature recognition approach to safety-critical applications by computing the probability of Hazardous Misleading Information (HMI) online [2]. *Joerger's* work serves as a cornerstone for the further research presented in this thesis on map integrity.

1.3 Contributions

This thesis presents a laser-based localization approach utilizing the techniques of geometric hashing. Since the aim of this thesis is to address the challenges of model-based recognition and fit within the framework of feature-based localization, careful analysis will be provided to show how these challenges can be overcome using geometric hashing. As part of this technique, a new approach to quantifying feature map symmetry will be introduced. This concept of obtaining such a metric identifies a small research gap within the community of model-based techniques. With prior identification of problematic regions within a model, recognition algorithms can appropriately analyze the risk of corresponding with the perpetrating features.

In addition to the challenges presented above, geometric hashing localization will also address several other challenges associated with real-time autonomous systems such as the "Kidnapped-robot" problem. A simulation using Ackermann-like kinematics was developed as an initial proof-of-concept. To verify the feasibility of the approach under strict time-demanding conditions, the algorithm was implemented into an existing localization pipeline running on a Mercedes E-class coupe. To summarize, this thesis contributes to the following key points:

- Development of a parallelizable laser-based data association technique immune to partial object occlusion and rotation
- An offline analysis framework to determine the integrity of feature maps
- An online risk reporting approach for feature associations paired with maximal error bounds
- Localization using efficient map/database memory footprints
- Integration of the data association algorithm into an existing localization framework requiring time-critical updates

1.4 Outline

The remainder of this thesis will transition through phases beginning with general theory and ending with a practical real world implementation. Chapter 2 will begin by introducing geometric hashing as it exists today within the research community. A simulated example will assist in demonstrating how the algorithm works as well as highlight the strengths and weaknesses under certain conditions. To close up the chapter, prior contributions utilizing geometric hashing will be presented and discussed. Chapter 3 will discuss in further detail one of the critical problems plaguing Model-based Recognition: Map Ambiguity. The process of feature extraction will be introduced briefly in order to generate feature maps. After providing an example of how prevalent map ambiguity can be, a deterministic approach to detecting symmetry among features using the geometric hashing framework will be presented. Chapter 4 will transition toward an implementation of geometric hashing as a positioning solution for laser-based ground vehicles. The algorithms that exist as part of the implementation will be explained along with several practical considerations. Chapter 5 will introduce the test environment used to generate results for the implementation. This will involve a description of the routes driven by the test vehicle, the on-board sensor suite that provided the data, and the generated requirements and conditions for the experiment to be weighed against. Chapter 6 will present and discuss results from the experiments designed in Chapter 5. Finally, Chapter 7 will conclude

this work by reflecting on geometric hashing as a possible localization solution for laser-based ground vehicles and present avenue for further research.

Chapter 2

Geometric Hashing

In order to clearly understand how geometric hashing can function as a valid real-time localization technique, a background of the theory behind the approach must first be presented. In general, the underlying concept of geometric hashing is a modified application of a multi-purpose algorithm known as a **hash function**. The primary directive of this genre of algorithms is to map arbitrarily-sized data to a fixed length, resulting in an indexable **hash code**. The modification of this algorithm to encode geometric information is what provides the technique with the model-based characteristics necessary for feature recognition. Akin to other model-based approaches, the procedure for geometric hashing is separated into two phases. The first phase, known colloquially as the Training Phase, employs a series of setup tasks that are performed offline. These steps calculate and build the database of models that are later used for recognition. The second phase of geometric hashing, known as the Recognition Phase, is conducted online and contains the model matching algorithm used for object recognition.

The contents of this chapter will expand further on the directive of hash functions and describe how they can be used together with model-based techniques. Over the next few sections, a derivation of geometric hashing will be presented to clarify how it seeks to overcome the four challenges plaguing model-based recognition as introduced in the previous chapter. After the derivation, a discussion of additional unique characteristics belonging to geometric hashing are provided within the context of localization. Finally, the chapter will close with a summary of several relevant applications of this technique within the research community.

2.1 Basis Formulation

The first challenge in model-based recognition, locating objects that have undergone geometric transformations, can be eliminated by re-defining each model as a collection of transformation invariant features. Geometric hashing easily accomplishes this task by defining a

geometric basis comprised of an arbitrary k -tuple of primitive features from a given model. In mathematical terms, k is called the cardinality of the basis and represents the number of elements that belong to a set. After this, the position of each remaining primitive feature in the model can be represented in the newly-defined geometric basis. Now, the position of each primitive will be preserved, even if the object matching the model is seen from a different point of view. Since the primitives here are definitively invariant to geometric transformations, they can henceforth be called **invariants**. This concept is easily demonstrated through the notion that each primitive is now defined relative to a location on the model instead of a global frame.

Before providing a specific example of this, it is important to note that there are several different classifications of geometric transformations:

- Isometric - preserves distances and angles,
- Similarity - preserves ratios of distances and angles,
- Affine - preserves parallelism between geometries.
- Projective - preserves collinearity between geometries.

With each class, a different formulation of the basis is required to grant transformation invariance of the same class. Since the scope of this work requires an exact (i.e. rigid) matching of primitive features, it is undesirable to allow scaling and shearing in transformations. Therefore, we will only investigate the derivation of the isometric, or rigid, transformation in this chapter.

Using a simple example, the steps to derive a geometric basis for the rigid case as stated in *Rigoutsos's* work will be restated [7]. Figure 2.1 depicts a model M consisting of several primitive features in two dimensions. The position vector of each primitive relative to the model origin is represented as $\vec{p}_i \in \mathbb{R}^2$, where $i = 1, 2, \dots, 5$. In many cases, additional information about each primitive, such as attributes or descriptors, are also contained in the model. To begin, first select any arbitrary k -tuple of features from the model. Given the desired constraints for rigid transformations, it is only necessary to select two features to define our basis (i.e. $k = 2$). Assuming the selection of \vec{p}_1 and \vec{p}_2 , the x-axis of the basis \vec{b}_x can now be defined as the vector

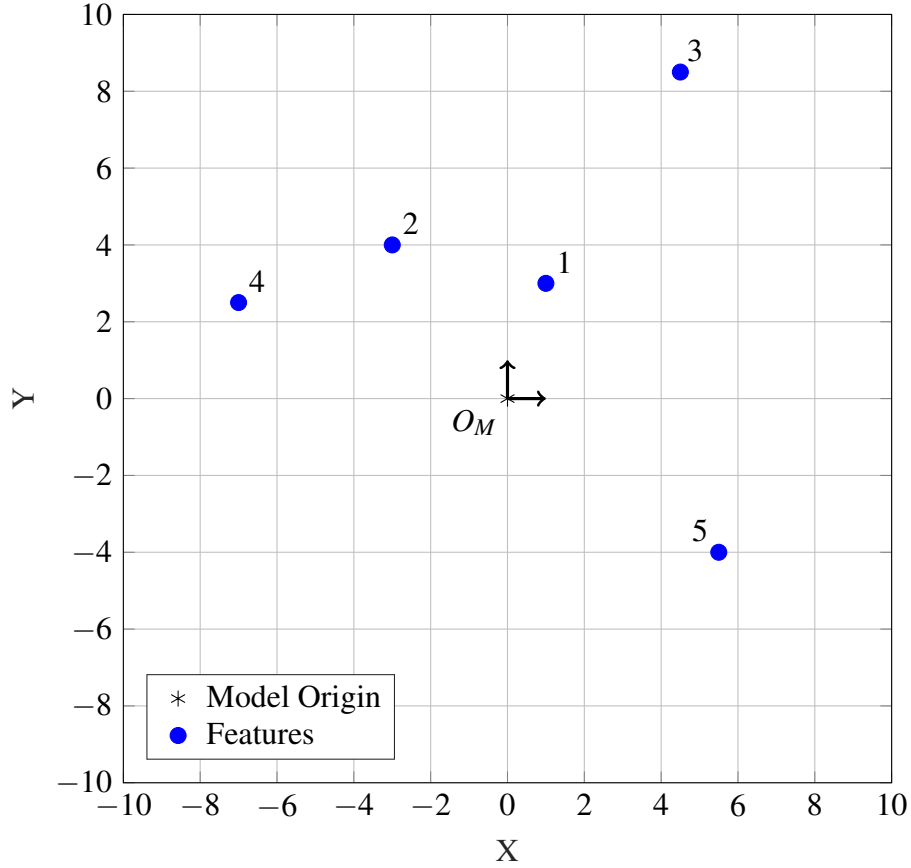


Figure 2.1: Model of extracted point features

from \vec{p}_1 to \vec{p}_2 . Likewise, an orthogonal vector \vec{b}_y can also be defined to represent a right-hand coordinate frame.

$$\vec{b}_x = \frac{\vec{p}_2 - \vec{p}_1}{\|\vec{p}_2 - \vec{p}_1\|} = \begin{bmatrix} \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \\ \frac{y_2 - y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \end{bmatrix} \quad (2.1)$$

$$\vec{b}_y = \frac{(\vec{p}_2 - \vec{p}_1)^\perp}{\|\vec{p}_2 - \vec{p}_1\|} = \begin{bmatrix} \frac{y_1 - y_2}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \\ \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \end{bmatrix} \quad (2.2)$$

The definition of the basis origin is subjective and can vary depending on the application. Nevertheless, for the simplicity of this derivation, the origin \vec{p}_o will be defined as the midpoint between \vec{p}_1 and \vec{p}_2 as shown in Figure 2.2a.

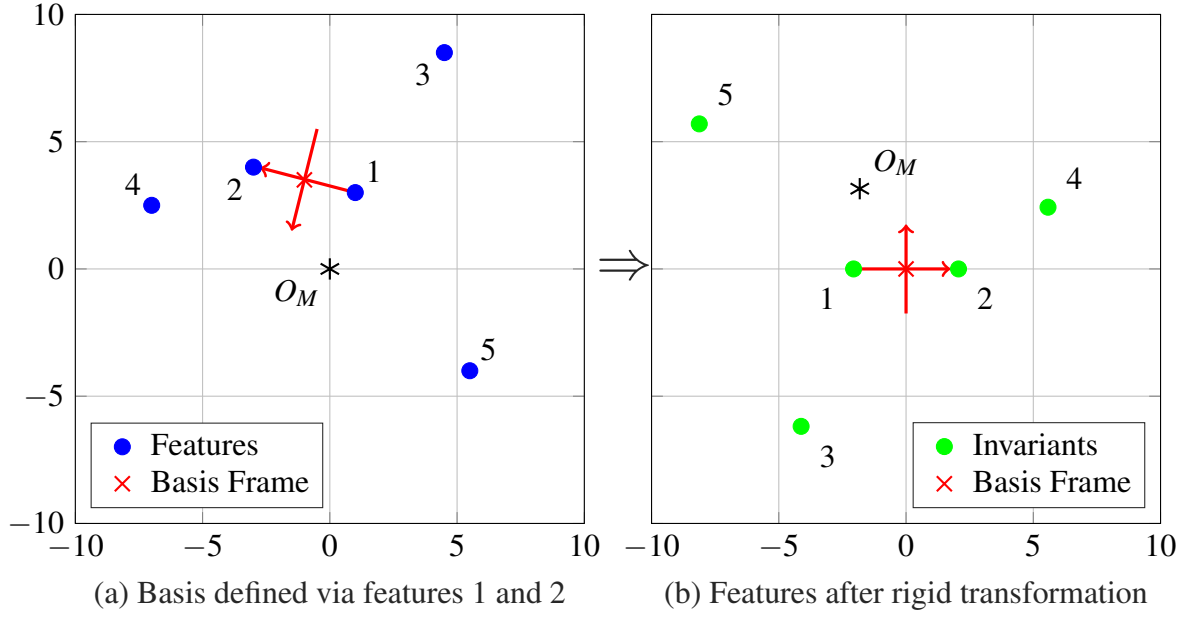


Figure 2.2: Creation of Invariants through Rigid Transformation

$$\vec{p}_o = \begin{bmatrix} x_o \\ y_o \end{bmatrix} = \begin{bmatrix} \frac{x_1+x_2}{2} \\ \frac{y_1+y_2}{2} \end{bmatrix} \quad (2.3)$$

Next, each of the remaining position vectors \vec{p}_3 , \vec{p}_4 , and \vec{p}_5 must be transformed into the newly defined coordinate frame. Since \vec{b}_x and \vec{b}_y are known, it is possible to re-define the position of each primitive as a linear combination of each basis vector scaled by some constants μ and ν . From this, Equation (2.4) is obtained. Solving Equation (2.4) for the constants μ and ν , the formulation for computing the position of rigid-invariant features in the basis domain is acquired. Now that each of the primitives are defined relative to the basis frame as shown in Figure 2.2b, their local positions will result in the same values even if the points from Figure 2.1 are seen from different viewpoints.

$$\vec{p}_i - \vec{p}_o = \mu \vec{b}_x + \nu \vec{b}_y \quad (2.4)$$

$$\begin{bmatrix} \mu \\ \nu \end{bmatrix} = \frac{1}{\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}} \begin{bmatrix} x_2-x_1 & y_2-y_1 \\ y_1-y_2 & x_2-x_1 \end{bmatrix} \begin{bmatrix} x_i - \frac{x_1+x_2}{2} \\ y_i - \frac{y_1+y_2}{2} \end{bmatrix} \quad (2.5)$$

2.2 Hashing Functions

The second challenge in model-based recognition, pertaining to occluded objects, is also adequately accounted for in geometric hashing. Using the formulation derived in Section 2.1, it is possible to collect all possible un-ordered k -tuple combinations of primitives for any given model. Mathematically, this is equivalent to computing the binomial coefficient of k for n features.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad (2.6)$$

By obtaining all possible bases and their respective invariants from a model, the likelihood that an object will remain distinguishable in the presence of occlusion increases significantly. For example, in the event that only one basis is created for a given model, it will become nearly impossible to recognize a corresponding object if one (or both) of the primitives used to create that basis is occluded. In other words, it is necessary to capture all possible bases for each model in order to decrease the chance of failure due to occlusion.

Although computing and storing all this information considers one problem, it creates several more:

- costly computational time,
- machine factorial limits,
- large data storage requirements and,
- searching large data.

For applications that contain a large database of complex models, the computational time required to cycle through all possible combinations could quickly become impractical. In addition, as the cardinality of the basis formulation (i.e. k) increases, the computational complexity will increase exponentially. Luckily, these two issues will not have any effect on our scenario since this step is performed offline and the formulation only permits rigid transformations of

points. However, after completing the database, there still exists a task of searching for the correct match. This issue contributes toward the necessity of the hashing component in geometric hashing.

As stated in the introduction of this chapter, geometric hashing utilizes a hash function to encode arbitrarily-sized information into easily searchable hash codes. This algorithm provides two major benefits to the application. First, using a hash function enables geometric information to be re-mapped to a deterministic value. In other words, a unique input to a hash function will always produce the same unique output. Here, a short example of how this concept adds value is presented. Since all bases contain rigid-invariant features, a possible corresponding geometry in a scene could produce a matching set of rigid-invariant features. In order to determine if any (or all) features match with the model, a possible approach could be to individually check each position of the invariants. Even with technology that exists today, this procedure would be too slow for real-time conditions and cost unnecessary resources. However, since the directive of the hash function always maps data to a deterministic value, matching the invariants simply becomes indexing to the correct hash code. The second benefit is the ability to encode comparatively large information into compact hash codes. This trait enables the ability to store large amounts of nearly-redundant information in a compact and efficient manner. The data structure in which hash codes are stored is called a **hash table**. Each code in the table is paired with a **bucket** and together referred to as a **hash entry**.

In order to ensure the hash function produces desirable results for recognition, it is often necessary to quantize the invariants over the basis domain. After this process, a series of discrete quantities are created containing the information of the invariants. Each quantity is called a **bin**. In some cases after quantization, it is possible for invariants to fall into the same bin. If left alone, this phenomenon will cause a fault in the algorithm called a **hash collision** which eliminates the ability to differentiate between the two invariants during recognition. More information about hashing collisions as well as the benefits and detriments of quantization will be discussed later. Since both the quantization parameter and the hash function carry a profound influence over the structure and content of the hash table, they are both typically reserved as implementation-dependent parameters.

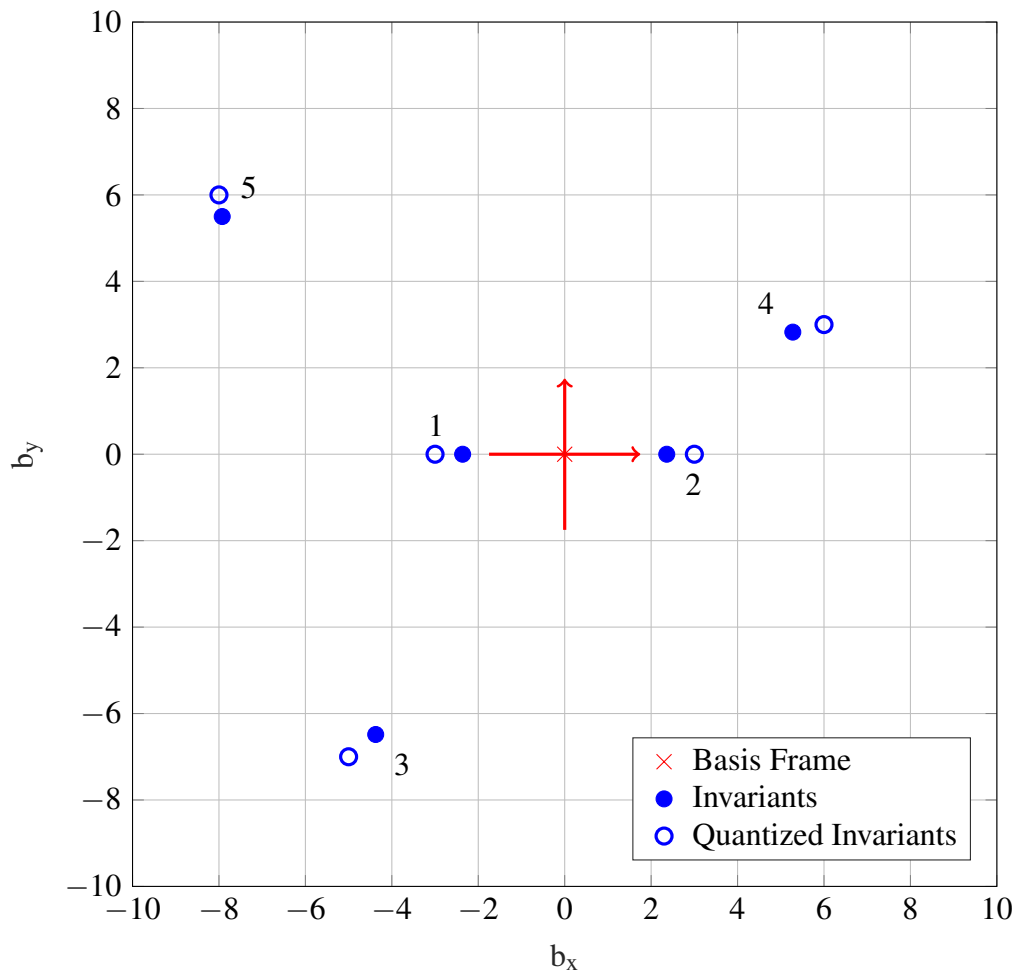


Figure 2.3: Quantization of invariants with $q_{pose} = 1$

Continuing on with the example from Section 2.1, it is now desired to apply a hash function to the set of invariants and construct the hash table. In the previous step, a set of rigid-invariant features \vec{I}_i^r , $i = 1, 2, \dots, 5$ belonging to the basis defined by \vec{p}_1 and \vec{p}_2 was obtained. Next, the positions of the invariants are discretized according to a quantization parameter q_{pose} as shown in Figure 2.3. Within this thesis, this parameter is also called the bin size.

Now, using selected information from the quantized invariants, a hash code can be computed for each invariant in the basis and stored in a indexable list of unique elements. For example, computing the hash value for the quantized μ and ν components of \vec{I}_5^r could be as simple as a conversion to a 32-bit, two's complement, hexadecimal hash string as shown in Equation (2.7). Along with the hash code, it is also required to save critical information about the invariant, such as the model and basis it belongs to as well as an identifier linking the invariant back to the model. This information is generally stored in the bucket of the newly created hash entry depicted in Table 2.1.

$$(\mu_5^q, \nu_5^q) = (-9, 6) \Rightarrow \text{0xFFFFFFFFFA00000006} \quad (2.7)$$

Table 2.1: Structure of hash entries

Hash Entry	
<u>Hash Code</u> $\mathbf{h}(\mu, \nu)$	<u>Bucket</u> $model, layer, feature$

After conducting this process for each invariant, it is then necessary to repeat the process for all subsequent bases feasible within the model. Each basis, as well as the information owned by it, is called a **layer** and is stored alongside the list of hash entries within the hash table. If any invariant should hash to an equivalent hash code belonging to a different layer or model, the new information is appended to the existing hash entry and will be reconciled during recognition.

The origin of the *layers* terminology stems from the matching procedure within the basis domain. For example, suppose after computing all possible bases for a model, it is desired to compare this collection of bases with another individual basis computed from a local scene.

One possible technique to visualize this comparison is to "stack" each basis on top of one another such that the origins overlap and the basis vectors remain aligned. Naturally, this concept has no practical meaning in the model frame since each basis represents a different physical location and orientation. However, considering the various bases in a stack enables an extremely fast computation to find all instances with matching invariant positions to our local basis. In this sense, each basis represents a "layer" in the stack, and those instances that contain matching invariants are listed as candidates for the correct association. A top-view example of this is shown in Figure 2.4 where each color represents a unique layer in the hash table. This analogy will be continued during the derivation of the matching procedure in the next section.

2.3 Invariant Matching

Now that a proper overview of hashing transformation-invariant features has been presented, a gentle introduction to the recognition step can be given. As mentioned earlier in this chapter, geometric hashing consists of two primary processing phases. The first, encapsulating most of the procedure presented thus far, is called the Training Phase and is typically conducted *a priori*. As such, the process of generating a hash table can be conducted in an offline state, so as to avoid the interference with other real-time machine resources and allow for the greater computational time required to produce high-resolution databases. In contrast, the second phase of geometric hashing is performed during real-time operation (i.e. online). The directive of this phase is to utilize the models stored within the *a priori* hash tables in order to match with and provide correspondence to their physical counterparts within a scene.

Again, continuing with the example from before, the general procedure for recognizing models from the hash table will be discussed. Suppose a scene S of the surrounding environment is captured via an arbitrary sensor and several features of interest are extracted as shown in Figure 2.5. The position vector of each extracted primitive relative to the scene origin (i.e. sensor position) is represented as $\vec{p}_j \in \mathbb{R}^2$, where $j = 1, 2, \dots, 12$. Assuming the model from Figure 2.1 exists somewhere within the scene, it can be concluded that there also exists some rigid transformation \mathbf{T}_r that will satisfy $\mathbf{T}_r M \in S$. In order to successfully match an object in the scene with our model, the transformation \mathbf{T}_r must be determined.

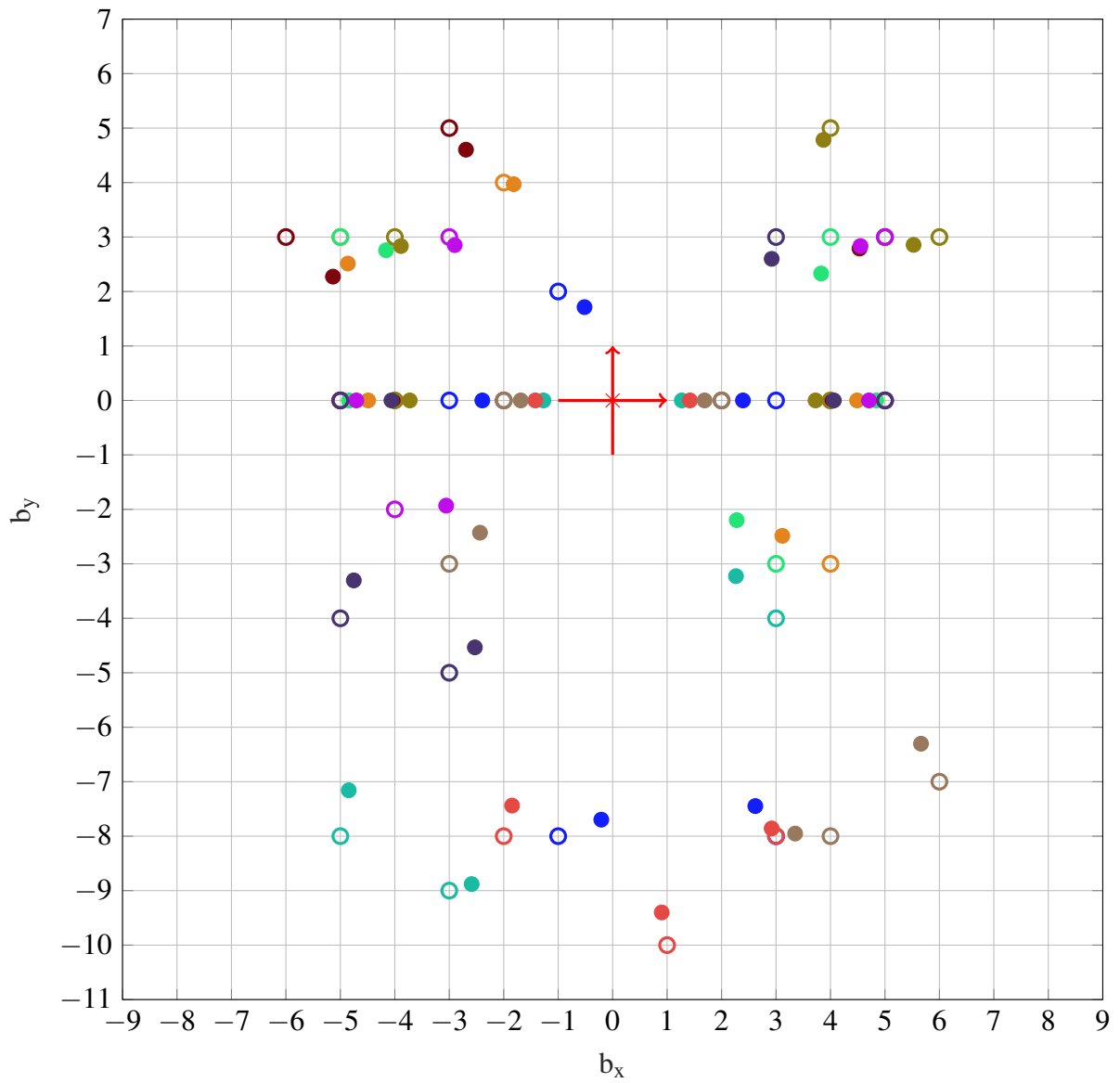


Figure 2.4: "Stacking" of bases to form layers

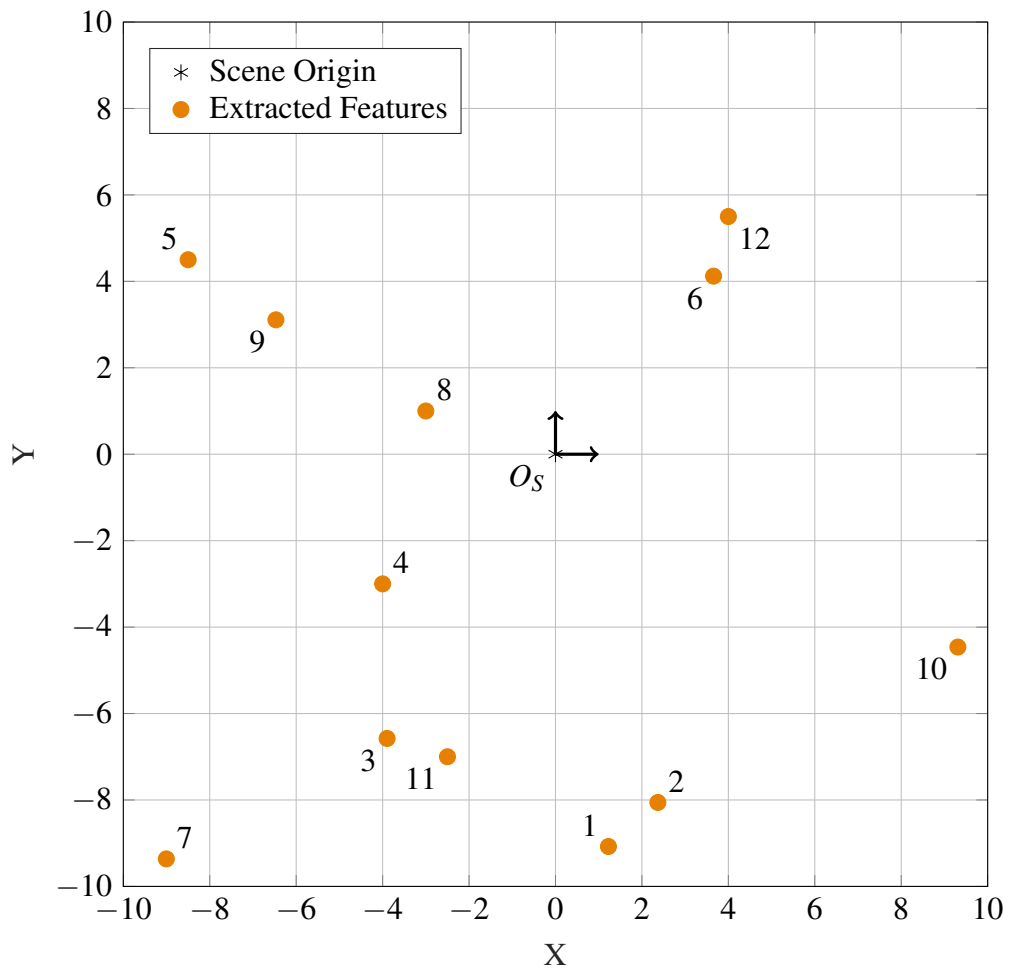


Figure 2.5: Scene of local environment containing extracted point features

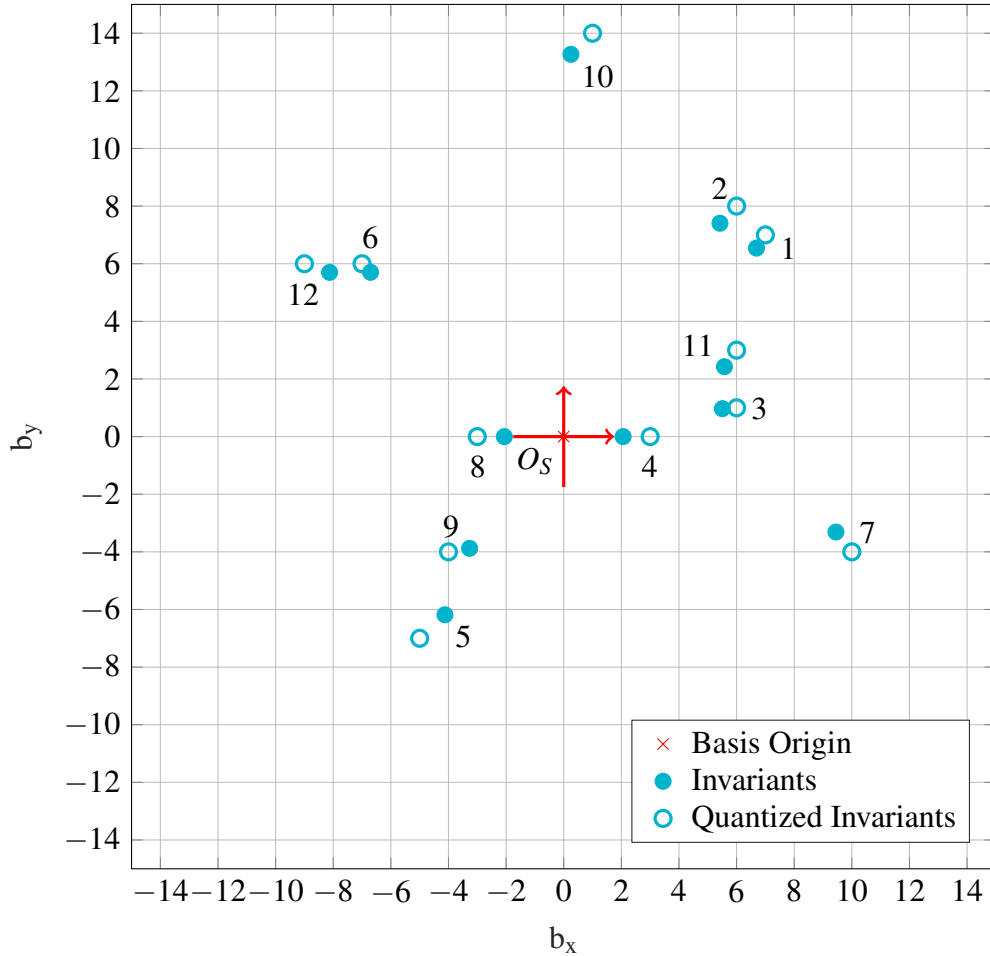


Figure 2.6: Rigid transformation of local scene

The initial steps for invariant matching follow closely to those performed during the offline procedure. First an arbitrary combination of k -tuple of features are selected from the extracted features in order to form a basis. Using the vernacular of model-based recognition, the creation of this basis with an attempt to match sets of features is called a **probe**. Just as before, a geometric basis is computed and all extracted features are transformed into the new domain as shown in Figure 2.6.

Next, the hash function is applied to each invariant and search for a matching hash code in the hash table. If a match is found, this indicates that an invariant from the corresponding layer is geometrically similar to the local scene. In order to acknowledge this, a vote is cast for each model-layer tuple that exists within the bucket of the hash entry. After cycling through all invariants within the basis and casting votes, a final tally of the results is performed. In an ideal

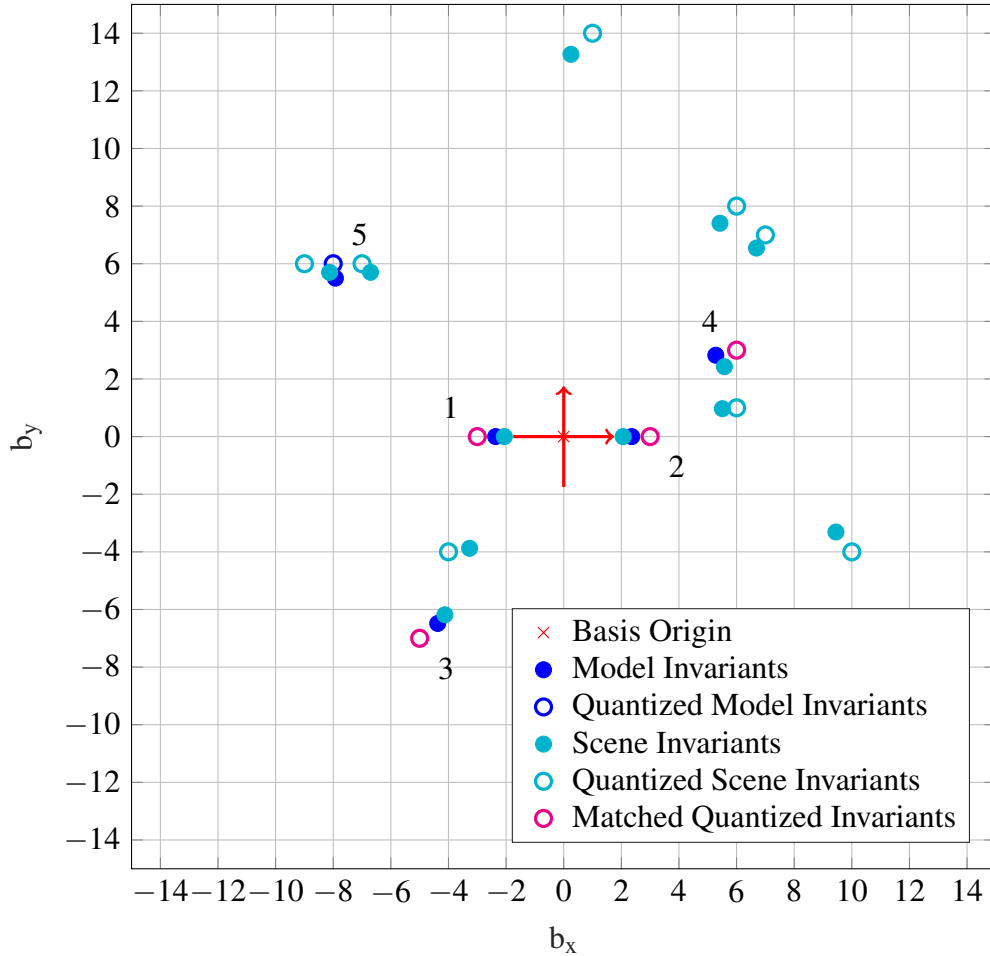


Figure 2.7: Recognition of model within the scene

scenario, the model-layer tuple receiving the most votes after the tally is deemed the match and the transformation \mathbf{T}_r can be obtained. With this information, the sensor position is now effectively localized with a model element. Since our example is relatively simplistic, it is easy to visually determine the position of the model within this scene as shown in Figure 2.7.

In more realistic scenarios which will be discussed in a later chapter, the recognition procedure is not as trivial and instead requires an additional verification step in order to determine the most likely association. In some cases, it may be necessary to conduct several probes in order to build a confidence parameter and provide reliable results.

To summarize, the concept of creating a geometric basis from all possible combinations of k -tuple primitive features can be thought of as similar to taking a snapshot of the same object from different angles. In a practical sense, this would significantly increase the probability

of recognizing that object in a unknown random environment. As it stands, this technique provides a quality appeal to several challenges in our list. Creating geometric bases alleviates issues with transformations, segmenting models into primitive features enables the recovery of partially occluded objects, and utilizing hash functions promotes efficient data storage and low resource expenditure. However, there are still several other characteristics to consider before praising this technique as a practical solution to the challenges.

2.4 Other Characteristics of Geometric Hashing

Since a comprehensive introduction of geometric hashing for general model-based recognition has been presented, it is now desired to understand several critical characteristics of the technique within the context of real-time localization conditions. This section will introduce several of these characteristics as well as highlight other works that have contributed to these topics.

2.4.1 Concurrent Processing

One major advantage of using geometric hashing in time-critical applications is the ability to utilize concurrency. Depending on the application, both the training and recognition phases can operate in a parallel fashion within their individual procedures. In the case of both phases, a set of geometric bases must be created and all remaining primitive features must be transformed into the new frame. In general, this process highlights two distinct steps that could be computationally parallelized.

The first, is the creation of the basis. Since each geometric basis belonging to a given model or scene is mutually exclusive, it is possible to compute each basis simultaneously. From Equation (2.6) presented earlier, the number combinations of features necessary to utilize all possible information from the model or scene can be determined. With the combinations known, each combination can be dispatched to the basis creation procedure on different processing elements (PE) if available. The second step that could be run concurrently is the transformation and hashing of each primitive. Since there exists no co-dependency among the features relative to the basis origin, each transformation and hash function could be performed

using parallel logic. However, as with all concurrent processing procedures, it is important to understand the relationship between parallel logic and the monolithic hash table. Since each of the concurrent tasks are communicating with the hash table, careful consideration must be given to avoid collisions between the tasks, and consequently, the loss of information.

An application of parallelization in geometric hashing is presented in great detail within *Rigoutsos's* dissertation [7]. In his work, geometric hashing using large databases of models in parallel on a Connection Machine is studied and implemented. This application serves as a perfect example for the necessity of computational concurrency when certain conditions are present. *Rigoutsos* has already stated that matching rigid-invariant features operates in $O(S^3Mn^3)$ time where S represents the number of bases created from online extracted features in the scene, M represents the total number of bases create from model features, and n represents the number of features in the model. Therefore, as the number of models and geometric complexity increases, the time required for recognition can quickly become impractical. By utilizing multiple processing elements and concurrent computation, the time required to detect matches among the multitude of models can be significantly reduced.

2.4.2 Sensitivity to Noise

At this point in the discussion, the performance of geometric hashing within the context of our specified challenges from Chapter 1 is promising and appears mathematically sound. However, all of the formulation thus far has been presented under the assumptions of ideal conditions. In most practical scenarios, noisy measurements provided from sensor data carry a heavy burden on reliability and efficacy. Geometric hashing is not immune to this burden. Within the overall procedure, there are two distinct sources of noise that could contribute to erroneous results: noise introduced via quantization and noise on the position of extracted primitives.

Looking first at the positional noise of extracted primitives, there has been a considerable amount of research dedicated towards this issue [19, 13, 14]. The findings show that even small amounts of noise or uncertainty in the model primitives has an amplifying effect on the transformed invariants. Looking ahead further, applying the hash function to noisy invariants could

result in incorporating incorrect hash entries in the table. In turn, this would cause numerous difficulties during recognition.

In order to mitigate the consequences of this problem, the noise on the extracted features can be characterized based on the sensors or algorithms that produced them and then an appropriate model can be constructed. By modeling the noise, a formulation of the resultant noise after the transformation can be incorporated into the hash table. Knowledge of this information enables a proper selection of quantization parameters which can also help alleviate stress on the recognition side. In a later chapter, knowledge of this issue will enable the derivation of a model for the noise on extracted model features allowing rigid transformations.

An additional technique could be to select a range, or **neighborhood**, of hash entries with in the hash table [14]. Naturally, even though the chance of missing correct hash values is reduced, this concept also increases computation during the recognition step. If the neighborhood of hash entries becomes too large, the recognition phase will quickly become saturated with candidates and the correct association may vanish below the noise floor.

During the quantization step, additional error is introduced into the data via round-off losses. Since each invariant is quantized into an appropriate bin, information on the true position of the invariant is lost when computing the hash value. This phenomenon introduces a sophisticated trade-off between large and small bin sizes. For example, quantization is necessary for geometric hashing because it enables the transition of "continuous" positional values into discrete locations. If the bin size is increased to a large value, more information will be lost, chances of a hash collisions grow, and an increasing number of bins will become overpopulated. The one good outcome is that larger bin sizes produce a smaller amount of bins to search through, but the advantages of this are effectively negated by overpopulation. On the other end, if the bin sizes are too small, the hash domain will become too sparse and data storage will become an increasing concern. Some evidence of this can be seen in Figure 2.7 where feature number 5 was not matched. In this case, the continuous space positions of the feature and detection were closer in proximity than in discrete space.

Apart from the selected bin size, another undesirable trait in the quantization process is the issue with boundary conditions. Assuming the invariants are quantized isometrically under

a user-defined schema, there will always exist scenarios where a boundary condition is met. In other words, if the schema involves rounding the position of each invariant to the lowest integer, those invariants that are already extremely close to an integer are considered to be on the boundary. This implies that the discrete value received from the quantization step has high potential to flip between bins if other noise is present.

2.4.3 Non-uniform Hash Distribution

Another noteworthy characteristic of geometric hashing is the distribution of hash codes over the valid region of the hash domain. In general, this trait is considered detrimental to the recognition step and increases the chance for incorrect associations to occur. Several prior research efforts have acknowledged this issue [19, 7] and have proposed various techniques to mitigate errors.

The root of non-uniformity among the hash domain is a direct result of the geometric spread of features within a given model. For example, looking back at Figure 2.4, it can be concluded that after quantizing the invariants into equally-spaced bins, the distribution of invariants across multiple layers is not uniform. This will result in certain hash entries receiving multiple layers, while others may receive none at all. Consequently, this phenomenon places a higher responsibility on the recognition step to decipher which layer is truly the correct association even though the candidates are close. Instead, a different approach could be to quantize the invariants into variably-sized bins following a user specific schema. This could alleviate the issue of multiple layers falling into the same bin, but the hash space will remain sparse and additional care must be taken to maintain the appropriate quantization in the recognition steps.

In order to maintain the simplicity of isometric quantization, a different approach must be taken to reduce the sparsity of the hash domain. One such approach involves a process called *rehashing* [7]. The idea rehashing is that the original distribution of invariants can be remapped into a uniform distribution, which will eliminate sparsity and reduce bins that are excessively populated. However, in order to do this, the original distribution must first be determined.

2.4.4 Detection of Feature Symmetries

The last characteristic to be discussed in this section is the ability to detect regions of symmetry among, or even within, the various models in the database. A major issue plaguing feature-based localization is the ability to distinguish certain features from others. In classical terms, this usually leads to the infamous "Kidnapped Robot" problem in mobile-base navigation. However, the procedure of geometric hashing provides an effective way to detect regions where this could be a problem.

Since the underlying procedure of geometric hashing transforms information into a model-based coordinate frame, the nature of this problem is slightly different from traditional applications. In a typical case, certain collections of features may resemble others contained within the *a priori* database, thus leading to ambiguities during association. However, for geometric hashing, the resemblance of features exists within the basis domain which provides an additional ability to detect similar geometries among models under transformation. Much of the ambiguity detection process is explored in *Hofstetter et al* [20] where a database of feature-based landmarks represented by geometric primitives is found to have a large number of symmetries. The steps behind this process will be introduced and described in Chapter 3.

Chapter 3

Detecting Ambiguities in Feature Maps

In Chapter 2, a comprehensive overview of geometric hashing for general feature recognition was presented. Moving forward, the contents of this chapter will transition toward a more applied focus using laser-based localization in mobile robot navigation. As stated previously in the definition of model-based recognition, models are considered to be any collection of extracted features obtained prior to recognition. In the classical use-case involving cameras and image processing, an arbitrary number of feature-defining pixels would be selected from a collection of images based on the needs of the system. Usually, these needs aim to provide sufficient positioning corrections to a localization solution while achieving a certain robustness to ambiguities within the working environment. However, since the techniques discussed later in this thesis utilize a laser scanner, a slightly different approach is needed to accomplish the task of identifying ambiguous model features and geometries.

Compared to monocular cameras, which provide a dense 2D projection of the scene, laser scanners produce sparse information within their field of view (FOV). One way they make up for the lack of dense information is by providing a metric of depth to each reported pixel, thus giving the consumer a 3D perception of the scene. A collection of laser range measurements, called a point cloud, and the feature models they compose within the geometric hashing framework are called feature maps. Similar to the image-based definition of a model, a feature map consists of an arbitrary collection of geometric primitives extracted from the desired environment. As explained in the previous chapter, obtaining geometric primitives has a distinct advantage of compacting information into smaller pieces. While this trait is beneficial to processing and storing information for later, it also creates difficulty during the recognition phase since information was lost from the compression. In this chapter, the main focus will be discussing and addressing this loss of information.

In the following sections, an overview of the problems resulting from map symmetry are given within the context of vehicle localization. In addition, a new approach to detecting symmetrical or ambiguous regions in feature maps using geometric hashing will be derived. Finally, the chapter will close with remarks of how detecting ambiguities can provide critical information to the localization pipeline in real-time applications.

3.1 Feature Extraction

Within any model-based localization framework, feature extraction plays a critical role in the overall effectiveness of the recognition solution. Naturally, features must first be reliably detected and extracted from a scene before they can be associated to a prior model. From this, in order to achieve the best results, it is imperative to choose features according to the operational environment of the system. For slow moving systems operating in tighter spaces, choosing complex models can provide unique, and consequently high integrity, updates since processing time may be less demanding. Faster moving systems requiring long-life operation will likely find greater success using more primitive models since they can be detected quicker and require less memory to store. The previous two examples highlight several factors to consider when choosing the correct model for a given system:

- Detection Rate - how fast can features be detected that *could* associate to the correct model
- Detection Repeatability - how likely is detection of the model across multiple instances/view-points
- Recognition Rate - how fast can features be associated to the correct model
- Model Similarity - how likely is it to confuse a detection for the wrong model
- Model Size - how much memory does storing a model require

Since the application of Geometric Hashing Localization presented in this thesis involves long-life driving in urban environments, greater focus is placed on choosing primitive models

to provide smaller feature sizes along with higher detection/recognition rates. The downside of this choice, which will be addressed in this chapter, is model similarity.

The feature extraction pipeline used predominately within this thesis is largely similar to the process introduced in the work by *Stefati et al* [21]. In their work, a novel approach to classifying cylindrical or pole-like objects from LiDAR data is studied and implemented. Given a point cloud representing a single 3D scan from a LiDAR, the ground plane is first classified and extracted according to the segmentation and binning method presented in *Himmelsbach et al* [22]. Once the ground plane has been identified, the remaining non-ground plane points are clustered and filtered to prevent over-segmentation. Since the cylindrical objects desired to be extracted are assumed to be vertical cylinders, the clusters can be tested against the basic circle equation shown in Equation (3.1) to determine an approximate radius and center point.

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \quad (3.1)$$

Reorganizing Equation (3.1) into a linear system of equations of the form $Ax = b$, a solution using each laser measurement z for the cylinder radius r and 2D center point (c_x, c_y) is provided in Equation (3.2) [21].

$$A = \begin{bmatrix} 1 & -x_1 & -y_1 \\ \vdots & \vdots & \vdots \\ 1 & -x_z & -y_z \end{bmatrix}, b = \begin{bmatrix} -(x_1^2 + y_1^2) \\ \vdots \\ -(x_z^2 + y_z^2) \end{bmatrix}, x = \begin{bmatrix} -r^2 + c_x^2 + c_y^2 \\ 2c_x \\ 2c_y \end{bmatrix} \quad (3.2)$$

To provide greater evidence of cylindricity, the circle fit test from Equation (3.2) is applied to various horizontal layers of points from the same cluster of points. In addition, an accuracy metric is obtained for each fit using the root mean square error e_{RMS} between the approximated radius and the distance from each point to the approximated circle center point [21]. This method for extracting features from LiDAR point clouds can be used to obtain primitive pole-like features during both a mapping or an online live run.

3.1.1 Map Generation

In order to produce the *a priori* information needed for the training phase of Geometric Hashing Localization, features must be extracted and stored in a map prior to running the localization pipeline. Since the possible use-cases for model-based recognition vary greatly in nature, the method chosen to generate the map is highly implementation dependent. It is beneficial however, to use a consistent approach when generating maps and extracting features to be matched to the map because it promotes the best chance to extract features that already exist within the map.

With this in mind, along with the feature extraction approach introduced above, the mapping procedure utilized in this thesis relies heavily on the methods from *Kümmerle et al* [23]. Maps are generated using a high fidelity mapping framework called *Surround View* presented in another work by *Sons et al* [24]. *Surround View* mapping utilizes a vision-only solution optimized for urban roadways with a diverse range of features. More information on *Surround View* will be discussed in Chapter 5.

Over a sequence of feature extraction events, detections representing the same object will likely overlap. Since this overlap is considered to be redundant information, these detections are first clustered via a proximity metric and merged into a single feature before getting added to the map. Clusters that do not have sufficient detections are considered to be outliers and are often discarded or filtered. This process helps to keep the map memory footprint small while utilizing the inherent confidence present for features with many overlapping detections.

3.2 Map Symmetries

The presence of map symmetries in mobile robot navigation causes enormous issues within the localization pipeline and greatly hinders system integrity and reliability. In a more classical sense, repeated instances of this phenomenon create similar conditions to the "Kidnapped Robot" problem from exploratory map learning. In these situations, the robot upholds a firm belief of the correctly localized position within the map, when in reality the evidence is considerably similar to another position and a lower degree of confidence should be held [25].

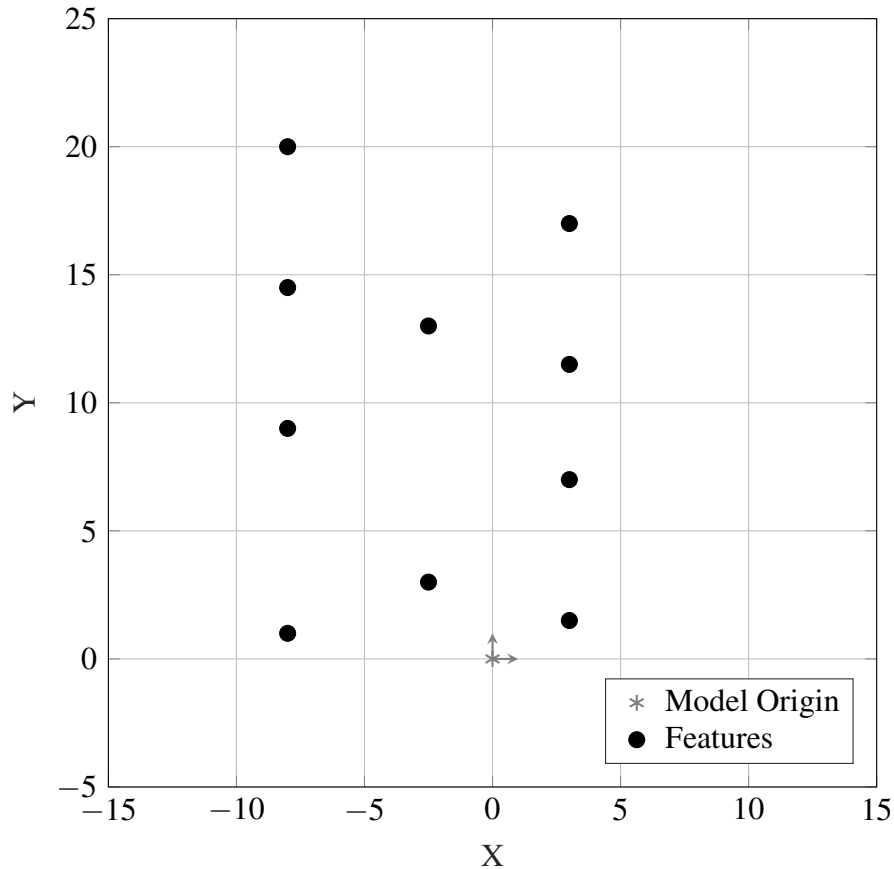


Figure 3.1: Sample feature map with symmetry

The consequences of this condition are extraordinarily detrimental to the integrity of each localization update and are generally undetectable through standard estimation techniques. As a result, the problem of map integrity identifies a subtle research gap in the field of probabilistic robotics using feature-based maps. Unfortunately, as with many other model-based localization techniques, the construct of geometric hashing is not immune to this phenomenon.

Recall in Chapter 2, it was introduced that geometric hashing operates on groups of features by taking advantage of the relative positioning of their members with one another. Because of this characteristic, the algorithm is inherently capable of observing instances where two groups of features are similar. With this metric known, additional filtering of information can be done to ensure that all possible evidence is considered before supplying the localization solution with a positional belief. Before diving into the steps behind this detection process, it is first important to understand what defines map symmetry within the context of feature maps.

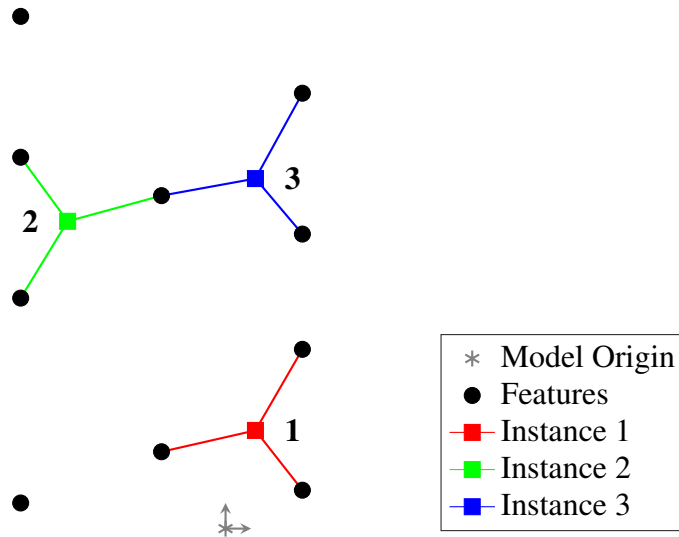


Figure 3.2: Highlighting a symmetric polygon in feature map

Suppose an arbitrary robot collects primitive feature information from a given environment to create a top-view map as shown in Figure 3.1. Before using this map for localization purposes, generally it is first desired to obtain information about its quality. For simplicity of the explanation, it is assumed that the feature positions within the map are noise-free and represent the true locations relative to the map origin. However, as mentioned before, noisy feature positions are not the only characteristic that contribute to the overall map quality. Looking at the positions of certain features, it can easily be seen that there exists subtle elements of periodicity among the features. In many cases, this is a clear indication that map symmetry is present and could result in the passing of misleading information to the navigation solution. Particularly within this example, the orientation of the points creates symmetry in higher dimensions as well. Since the feature map is defined for \mathbb{R}^2 space, only lines, curves, and other 2D polygons need to be considered as possibly similar or geometrically congruent.

As shown in Figure 3.2, there are 3 instances of a 3-sided polygon sharing geometric congruence. This example is particularly egregious because the instances all share the same

scale. As an example of how this phenomenon could negatively impact localization, imagine the robot is placed in the map such that its scene (*i.e.* perceptual field of view) contains only the three primitives that make up a single instance of the congruent polygon. The robot would use these features to match with the model and find 3 possible solutions. Assuming naivety, the robot has a 33% chance to select the correct position. With the techniques described in the following section, geometric hashing enables these instances of map symmetry to be detected *a priori*, thus providing a qualitative value to feature maps [20].

3.2.1 Detecting Similarities

Detecting all possible congruencies within a feature map using geometric hashing is a two phase process. The first phase involves an exhaustive search through all possible combinations of the layers stored in each bucket from the hash table. Because this first phase is known to generate duplicate information, the second phase is then responsible for consolidating all the evidence gathered from the first phase into a concise format. For maps with a large amount of features or poor geometry (*i.e.* dense maps with sequential feature patterns), the search performed in the first phase can quickly become infeasible. To make this more feasible, several constraints can be applied to place bounds on the search and reduce the overall computational complexity of the algorithm. While some of these constraints are considered implementation dependent, others, such as setting a maximum factorial size, apply generically to almost all applications and will be discussed further in this section.

To begin the first phase, it is necessary to look into the buckets stored at each hash value in the table independently. For any bucket that contains more than one layer, there exists evidence of similar geometry between each basis frame that defines those layers. Proof of this observation is inherent due to the nature of the recognition phase. Since all recognition operations are conducted in the basis domain, any invariant that shares the same coordinates (*i.e.* the same hash key) as an invariant in another basis necessitates that the two bases are geometrically similar. In order to visualize this concept, refer back to Figure 2.4 in Chapter 2. In that figure, all bases are superimposed onto a single grid and aligned accordingly in the basis

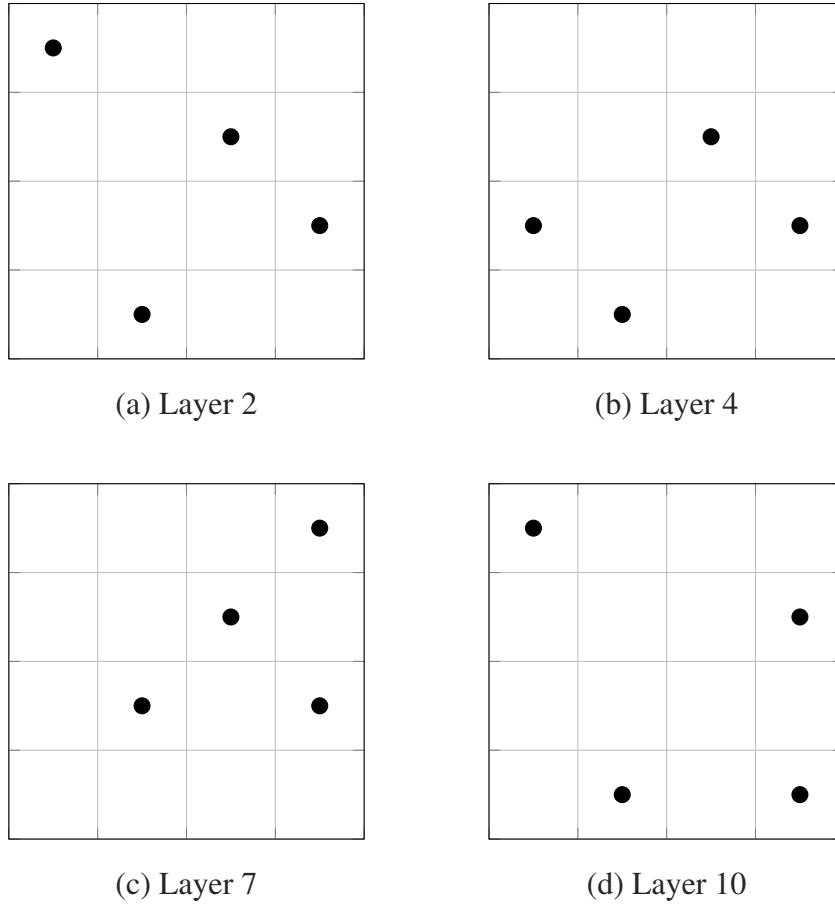


Figure 3.3: Layer comparison in the basis domain for bucket b

domain. A similarity can be visualized at the coordinate $(3,-8)$ where two points from different bases quantize to the same value.

Within each bucket containing two or more layers, all combinations of two layers are computed using the binomial coefficient formula introduced in Equation (2.6). Then, each combination is compared to determine if any invariants are congruent. This process is best demonstrated by an example. Suppose an arbitrary bucket b from a given hash table contains the set of layers $L_b = \{2, 4, 7, 10\}$ as shown in Figure 3.3. Using all possible unordered 2-tuple permutations of the layers, the list of unordered combinations is $(2, 4)$, $(2, 7)$, $(2, 10)$, $(4, 7)$, $(4, 10)$, $(7, 10)$.

Comparing layers 2 and 4 for similarities, it can be seen that 3 invariants are geometrically congruent between the layers. This subset of features is called a **similarity set**. In order to fully capture this information for use during the recognition phase, it is also important to consider

all possible k -combinations of this set. Without this consideration, there is risk that ambiguities will remain undetected if only a subset of the ambiguous features are present online within our scene. Using the total number of congruent invariants between the layers n , the number of similarity sets N_S is determined by Equation (3.3).

$$N_S = 1 + \sum_{k=1}^{n-1} \frac{n!}{k!(n-k)!} \quad (3.3)$$

Careful consideration must be given for layer combinations that produce large amounts of congruent invariants (*i.e.* n is large). The binomial coefficient equation utilizes a factorial operation which can produce numbers larger than standard machine address spaces. In addition, this introduces an $O(n!)$ computational complexity. Applying Equation (3.3) to our example shows that we should expect 7 similarity sets. A chart illustrating these sets organized by k is shown in Table 3.1.

Table 3.1: Generated Similarity Sets for layers 2 and 4

	Layers 2&4			...
...
3 Vertices				...
2 Vertices				...
1 Vertex				...

3.2.2 Forming Constellations

After indexing through all layer combinations in a bucket, a comprehensive list of congruent features is obtained within the basis domain. Since this list could potentially contain a large amount of redundant information, it is prudent to consolidate all duplicate instances of similarity sets. A similarity set is considered to be a duplicate if and only if each invariant within the set is congruent to an invariant in another set and both sets contain the same number of invariants. After consolidation, any unique set of invariants is referred to as an **ambiguous constellation**.

To begin this second phase, each row of Table 3.1 is searched for unique instances. In addition to the geometric pattern of the features, each instance of a unique similarity set also holds statistical information about its prevalence within the map. As an example, if the same similarity set appears twice in Table 3.1, then it should be expected that two instances of this constellation of features are present in the map. Since positional information about each similarity set is available, the relative transformations between each occurrence within the map can also be computed. Before doing so, it is desired to define a reference datum that is geometrically consistent between all occurrences of a similarity set. This ensures an unbiased result when converting positional information from the basis domain back to the map. A good candidate that satisfies this requirement is the geometric centroid of the invariants.

In order to compute the geometric centroid, the coordinates of each invariant in the map frame (x_i, y_i) for $i = 1, \dots, k$ can be treated as a vertex of an k -sided Euclidean polygon. Then, the coordinates of the polygon's geometric centroid in the map frame (C_x, C_y) can be computed using Equations (3.4-3.5) where the area A of the polygon is determined in Equation (3.6).

$$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \quad (3.4)$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \quad (3.5)$$

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \quad (3.6)$$

Notice, the area computed in Equation (3.6) can produce a negative value. In order to produce a valid area from Equation (3.6), the polygon must be categorized as a simple polygon (*i.e.* not self-intersecting). Since the polygon is composed of vertices, we can force a non-intersecting condition by sorting the vertices such that they follow a counterclockwise orientation. While the exact sorting method to perform this task is typically implementation dependent, it is only required that the approach produces repeatable results when subject to the noise on the map features. One such technique used in this thesis is to sort by the 4-quadrant angle value between the map positive x-axis and a 2D ray from an arbitrary point inside the polygon to each vertex. In the majority of cases, a point inside the polygon can be obtained from the pseudo-centroid (C'_x, C'_y) as provided in Equations (3.7-3.8).

$$C'_x = \frac{1}{k} \sum_{i=0}^k x_i \quad (3.7)$$

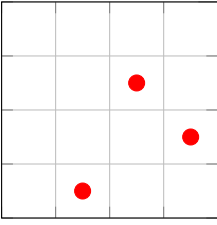
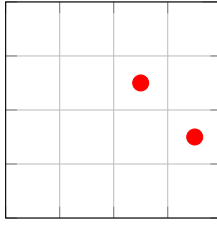
$$C'_y = \frac{1}{k} \sum_{i=0}^k y_i \quad (3.8)$$

Continuing the example of bucket b from Section 3.2.1, Table 3.2 illustrates the results of phase 1 and 2 of detecting congruencies where δ and θ represent the magnitude of the translational and rotational offsets between each occurrence respectively. Once all similarity sets are recorded for each layer combination within a bucket, this process is repeated for each remaining entry in the hash table.

3.3 Insight on Map Integrity

Obtaining the geometric transformations between all instances of an ambiguous constellation provides an upper bound on the resultant error should an incorrect instance of the constellation be chosen during recognition. Although this may not directly provide evidence to help choose the correct association, it enables each match to be paired with a risk metric as well as the maximal error if the wrong association is chosen. This can provide a downstream verification step critical information to decide whether to use the match or not.

Table 3.2: Ambiguous Constellations for bucket b

	Constellation 1	Constellation 2	...
Ambiguous Points			...
Occurrences	2	3	...
Transform Data	$\delta_{1,2} = 0.5$ m $\theta_{1,2} = 0.02$ rad	$\delta_{1,2} = 0.32$ m $\theta_{1,2} = 0.01$ rad $\delta_{1,3} = 0.76$ m $\theta_{1,3} = 3.14$ rad $\delta_{2,3} = 0.58$ m $\theta_{2,3} = -3.14$ rad	...

An additional metric available to provide insight about certain regions of the map is to generate a heat map for both map feature density and any ambiguous constellation occurrences. In regions of the map where there exists a high density of features there is a greater chance that the matching step will have sufficient information to produce an association. In contrast, areas that have very few features can be marked as low-confidence zones during recognition. Similarly, map regions contain a high density of ambiguous constellations will also be problematic. Because all of this information is known ahead of time, the system should expect the likelihood of misleading information to increase when traversing through these areas. Additional information regarding the data structures used to implement the theory presented in this chapter and Chapter 2 can be found in Appendix A.

Chapter 4

Geometric Hashing in Localization

Within this thesis, previous chapters have introduced several components of Geometric Hashing individually, but little practical discussion using these pieces together to aide a mobile robot has been provided. The complete procedure for Geometric Hashing consists of three phases: a Training Phase, a Screening Phase, and the Recognition Phase. Phases one and three of this process were introduced in an abstract form in Chapter 2 and phase two was defined in Chapter 3. In this chapter, the underlying procedures and algorithms for geometric hashing in laser-based localization will be presented. Each phase of geometric hashing will be revisited in a more condensed, procedure-focused form surrounded by the prerequisite steps to provide a complete picture of the *a priori* and online workflows. Lastly, several implementation-dependent nuances not previously mentioned in the other chapters will be shared.

4.1 Offline Processing

As mentioned previously in Chapter 2, the offline processing phase utilizes a collection of extracted feature models. For Geometric Hashing Localization, this collection is called a feature map and the generation of these maps using extracted features from 3D lidar measurements were covered in Chapter 3. Although numerous geometric primitive types were presented, this implementation of Geometric Hashing Localization will focus mainly on pole-like features. Recall, pole-like features, henceforth called cylinders, are represented in the map via two 3D Cartesian coordinates, indicating the top and bottom of the cylinder, as well as the radius. Using the two coordinates, several auxiliary descriptors can be determined such as cylinder height, length, azimuth, and tilt (elevation). Descriptors serve as an additional attribute to disambiguate potentially similar features during recognition and play a critical role when choosing from several candidates. One additional descriptor not mentioned in the prior list is the midpoint ground

projection point. The midpoint ground projection point is a special descriptor that enables the cylinder's positional information to be condensed into a single 2D point. This attribute is used to hash the positions of the cylinders into the hash table during the training phase.

4.1.1 The Training Phase

After obtaining the feature map for the intended operational environment, the training phase must begin by constructing a hash table. As mentioned briefly in Chapter 2, the hash table contains a list of hash entries, where each entry contains a mapping of hash codes to their respective buckets. In general, the buckets should contain all pertinent information required for the recognition phase. For geometric hashing, this boils down to each layer index that contains an extracted feature with this hash value. Under noisy conditions, it is also possible to store hash values belonging to the neighboring quantized locations around the hashed location. Further details on this will be provided later in this chapter.

In addition to the hash entries, the hash table also contains a database of all the basis layers indexable by a sequential identifier. The nature and ordering of these identifiers is typically determined by the structure of the feature map. Each layer in the database contains various meta information that establishes a relation between the real world and the basis domain. One critical component of this is a list of map feature identifiers that link each invariant in the layer (*i.e.* in the basis domain) back to an extracted feature in the map.

4.1.1.1 Basis Parameters

Before generating the hash table, there are several tuning parameters that can be used to lend flexibly to various operating conditions and environments. The most influential of these parameters is one that has already been mentioned in previous sections, the bin size q_{pose} . The bin size most heavily affects all three phases of Geometric Hashing because it determines the resolution of the hashing process. If a large bin size is chosen, there exists a higher likelihood of hashing collisions as more precision is lost during quantization. Although this will likely reduce the overall size of the hash table, it will have a negative impact on the robustness of the recognition phase due to information loss. Alternatively, choosing a bin size that is too

small will create the maximal amount of hash values with minimal collisions. This will give the hash table a larger memory footprint, but will better enable the recognition phase to select the best candidates in dense feature areas. However, choosing a bin size that is smaller than the average positional uncertainty envelope will likely lead to poor performance during recognition due to features "bleeding" into neighboring bins at the quantization step. More analysis of how altering the bin size affects performance will be provided later in this thesis.

Another pair of parameters focus heavily on limiting the scope of possible basis layers during the layer generation step. These are the maximum basis pair separation b_{limit} and the invariant inclusion radius r_{incl} . The basis limit establishes a rule that no bases can be created from a combination of features whose distance apart is greater than a set value. Similarly, the inclusion radius dictates that no hash values shall be created from invariants in a basis whose distance from the basis origin are larger than a set value. These parameters help to address the issue that the mobile platform is only able to see features within a set field of view during recognition. Since basis layers can only be generated if both points of the basis pair are seen, it is highly unlikely that any information larger than these parameters would be useful online. Due to the nature of the layer generation algorithm a limit can be established to enforce this as shown in Equation (4.1):

$$b_{limit} \leq r_{incl} \tag{4.1}$$

The last parameter, the scaling factor s_{trans} , affects the type of geometric transformation that is permissible during the layer generation step. For scaling factor values $0 \leq s_{trans} < 1$, similarity transformations are permissible. To force rigid transformations only, s_{trans} should be set to 1.

4.1.1.2 Quantization and Hashing

As mentioned earlier in Chapter 2, the hashing function selected to hash the invariants is typically implementation dependent. In general cases, the hash function only needs to reasonably guarantee unique hash values for the operational range of input values. For the implementation of Geometric Hashing Localization presented in this thesis, the range space is bounded by q_{pose} and r_{incl} . Because of this, the hash function was derived as a function of these two parameters as well as the quantized positions of the invariants.

The quantization method can be imagined by overlaying a grid structure onto the continuous space in the basis domain. Assuming a perfectly square shape (*i.e.* length and width contain the same number of cells), the number of cells in the grid N_g is shown in Equation (4.2).

$$N_g = \left(\frac{2r_{incl}}{q_{pose}} + 1\right)^2 \quad (4.2)$$

Similarly, the grid cell containing the basis origin O_g can also be obtained via Equation (4.3).

$$O_g = \text{ceil}\left(\frac{N_g}{2}\right) \quad (4.3)$$

Given that each cell is also square and sized via the q_{pose} parameter, continuous space points \vec{p} can be quantized to fit in a grid coordinate via Equation (4.4) where \vec{p}^q is the discrete position.

$$\vec{p}^q = \text{sign}(\vec{p}) * \text{ceil}\left(\frac{|\vec{p}|}{q_{pose}}\right) * q_{pose} \quad (4.4)$$

To reduce \vec{p}^p from \mathbb{R}^2 to a singular value, the hash function provides the method to map the discrete grid coordinate to a cell index in the grid. This is shown in Equation (4.5) where S_x and S_y are both scaling factors that provide the boundary constraints from q_{pose} and r_{incl} mentioned earlier.

$$\begin{bmatrix} S_x \\ S_y \end{bmatrix} = \begin{bmatrix} \frac{1}{q_{pose}} \\ \frac{1}{q_{pose}} \left(\frac{2r_{incl}}{q_{pose}} + 1\right) \end{bmatrix} \quad (4.5)$$

These scaling factors are then used in Equation (4.6) where h is the resultant hash value and μ^q and v^q are the vector components of p^q from Equation (4.4).

$$h = O_g + \text{round}(S_x \mu^q) + \text{round}(S_y v^q) \quad (4.6)$$

4.1.1.3 Collision Filtering

Before performing the hashing step, it is prudent to filter features that could produce hashing collisions from the map. As mentioned earlier in Section 2.2, hashing collisions result in two inherently different sources of information becoming indistinguishable from one another after the hash function is applied. In the case of geometric hashing, this means that two features are close enough that they will hash to the same value due to quantization errors. Since the quantization parameter q_{pose} is an input to the hash table parameters and the cells are known to be square, the minimum feature separation metric d_{min} is known as shown in Equation (4.7).

$$d_{min} = q_{pose} \sqrt{2} \quad (4.7)$$

After identifying all 2-tuple combinations of features needed to create bases, each combination should be checked to see if the threshold from Equation (4.7) is violated. For each violation, there are a few options for how to resolve the collision and each have varying degrees of severity. In the most strict case, all features involved in each hashing collision are removed from the map. This strict filtering method is particularly useful when the magnitude of positional uncertainty for a given extracted feature overlaps with the other colliding feature(s). When the aforementioned condition is met, there exists an increasing probability that the two extracted features correspond to the same object and no definitive conclusion can be drawn as to which, if either, features accurately represent the object's true position.

In certain situations however, such as within sparsely populated map regions, removing all instances of colliding features can create poor feature geometry or dispose of potentially helpful information. In these cases, a moderate collision filter can attempt to disentangle certain collisions by throwing away features that belong to more than one collision. For example, given

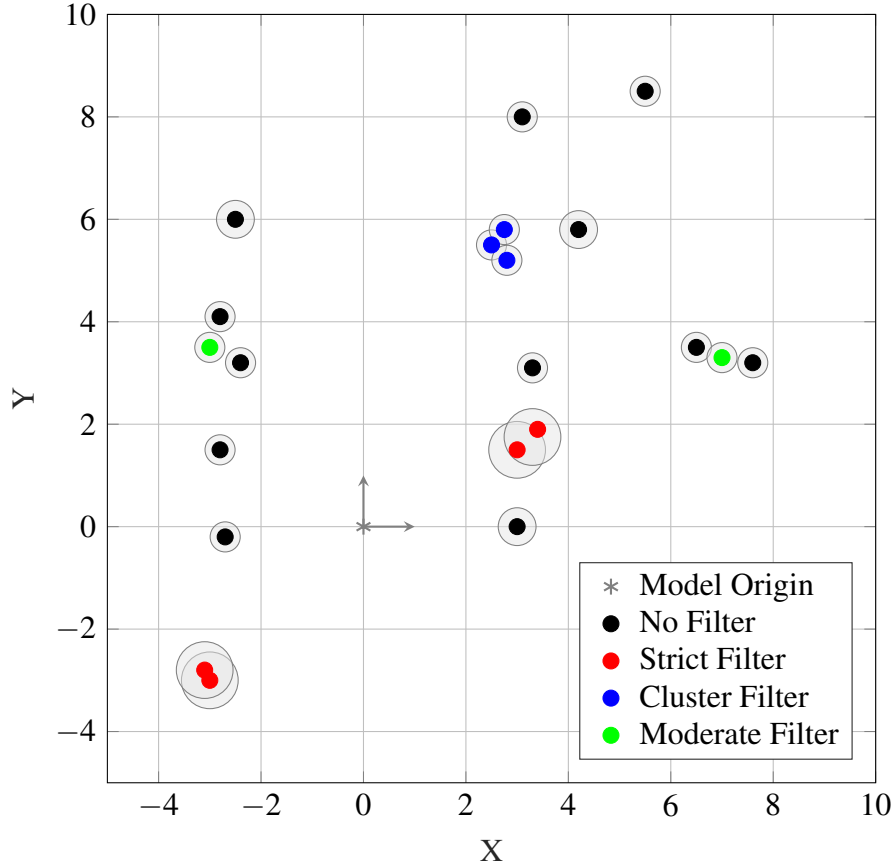


Figure 4.1: Hash Collision Resolution Methods - Bin Size: 0.5m

three extracted features with identifiers m_i , where $i = 1, 2, 3$, suppose that m_1 collides with m_2 and m_2 collides with m_3 . If m_1 does not collide with m_3 and all three features have non-overlapping positional uncertainty, then the optimal resolution action, in order to maintain the most information, is to remove m_2 . In another case, if m_1 and m_3 also collide, the positions of all three extracted features could be clustered to form a single map feature with an inflated positional uncertainty.

An example of each filtering technique is provided in Figure 4.1 given an example scene and q_{pose} of 0.5 meters. For each extracted feature, the covariance ellipse representing the positional uncertainty is represented by a translucent light gray circle surrounding the feature. Further inspecting Figure 4.1, a case where strict filtering is ideal is shown in two instances where a pair of red features with large uncertainty are close in proximity. The result of this filter removes all red features from the map. A case for moderate filtering is shown in two

instances where three features with non-overlapping uncertainties are in close proximity. In each instance, the feature shown in green is marked for removal from the map. Lastly, a case for cluster filtering is shown in one instance where three blue features are depicted. Since these features all have overlapping uncertainties, the recommended approach is to merge the cluster into a single feature and inflate the uncertainty to cover the total region of uncertainty before the merge.

4.1.1.4 Algorithm: Training

In order to consolidate all the steps for the training phase presented in Chapter 2 and earlier in this chapter, Algorithm 4.1 lays out a process flow in pseudo code. The function "CREATE TABLE" requires the complete set of features M from the map and returns a hash table H and database of layers L .

Algorithm 4.1 Training Phase

```

1: procedure CREATE TABLE( $\vec{m}_i, \vec{m}_x, \vec{m}_y$ )           ▷ Create hash table from set of features  $M$ 
2:    $l \leftarrow 0$ 
3:    $C \leftarrow nchoose2(length(\vec{m}_i))$                  ▷ Returns a  $c_x \times 2$  matrix of combinations
4:   for  $i \leftarrow 1, length(C)$  do                   ▷ Loop through combinations
5:      $\vec{c} \leftarrow (C[i][1], C[i][2])$ 
6:      $\delta \leftarrow distance(\vec{m}_x[c_x], \vec{m}_y[c_y])$    ▷ Get the scalar distance
7:      $\theta \leftarrow orientation(\vec{m}_x[c_x], \vec{m}_y[c_y])$  ▷ Get the angle of basis w.r.t. model frame
8:     if  $\delta < b_{limit}$  then                         ▷ Ensure basis pair are within distance threshold
9:        $l \leftarrow l + 1$ 
10:       $\vec{o}_l \leftarrow midpoint(\vec{m}_x[c_x], \vec{m}_y[c_y])$    ▷ Average the position to get basis origin
11:       $R \leftarrow rotmat(\theta)$ 
12:      for  $j \leftarrow 1, length(\vec{m}_i)$  do             ▷ Loop through all map features
13:         $\vec{p} \leftarrow (\vec{m}_x[j], \vec{m}_y[j])$ 
14:         $\vec{t} \leftarrow \vec{p} - \vec{o}_l$ 
15:        if  $r_{incl} < distance(t_x, t_y)$  then         ▷ Ensure point is within inclusion radius
16:           $\vec{b} \leftarrow transform(\vec{t}, R, s_{trans}/\delta)$    ▷ Transform point into basis frame
17:           $H \leftarrow hash(\vec{b}, q_{pose}, l)$            ▷ Quantize and hash the transformed point
18:           $\sigma \leftarrow computeCovar(\vec{b}, \delta)$      ▷ Compute transformed covariance ellipse
19:           $L \leftarrow updateLayerInfo(j, \vec{b}, \sigma)$    ▷ Add helpful info to Layers database
20:        end if
21:      end for
22:    end if
23:  end for
24:  return  $H, L$ 
25: end procedure

```

4.1.2 The Screening Phase

Returning to the discussion of notable contents within the hash table, the layers may also contain reference information to any ambiguous constellations present within that basis. Typically, this information is stored in a similar database structure elsewhere in the hash table where each instance is given a sequential identifier. The detection and formulation of these ambiguous constellations is performed offline using the results of the Training phase as a prerequisite. Because of this, the intermediate process of analyzing the map is appropriately named the screening phase. For the implementation of Geometric Hashing Localization presented in this work, the Screening phase begins exactly as described in Chapter 3 with one minor addition addressed below.

4.1.2.1 Congruent Bases

Certain geometries of features can create a troublesome scenario where all quantized invariants from one basis (excluding the basis pair) are identical to those in another basis. This phenomenon is caused by **basis congruency**. The consequence of this issue creates a large number of ambiguous constellations in the hash table that should not be considered ambiguous. The explanation of this is best demonstrated by an example.

Consider a scenario where a group of four features are aligned in one dimension and equally spaced in another as shown in Figure 4.2. When a basis is constructed by combining points (5,3) and (8,7), notice that the resulting midpoints (*i.e.* basis origins) will coincide at the same location relative to model origin. After transforming the points into the basis domain, the invariants within each basis will be perfectly geometrically congruent as shown in Figure 4.3. Since it is known that the model identifiers that correspond to these invariants are the same between the two bases, this is not considered to be ambiguous even though it meets the necessary criteria. This phenomenon can be detected and avoided by checking the difference in position of any two basis origins in the world frame. If the difference is less than a congruency tolerance threshold, t_{cong} , and all the model identifiers match between the two

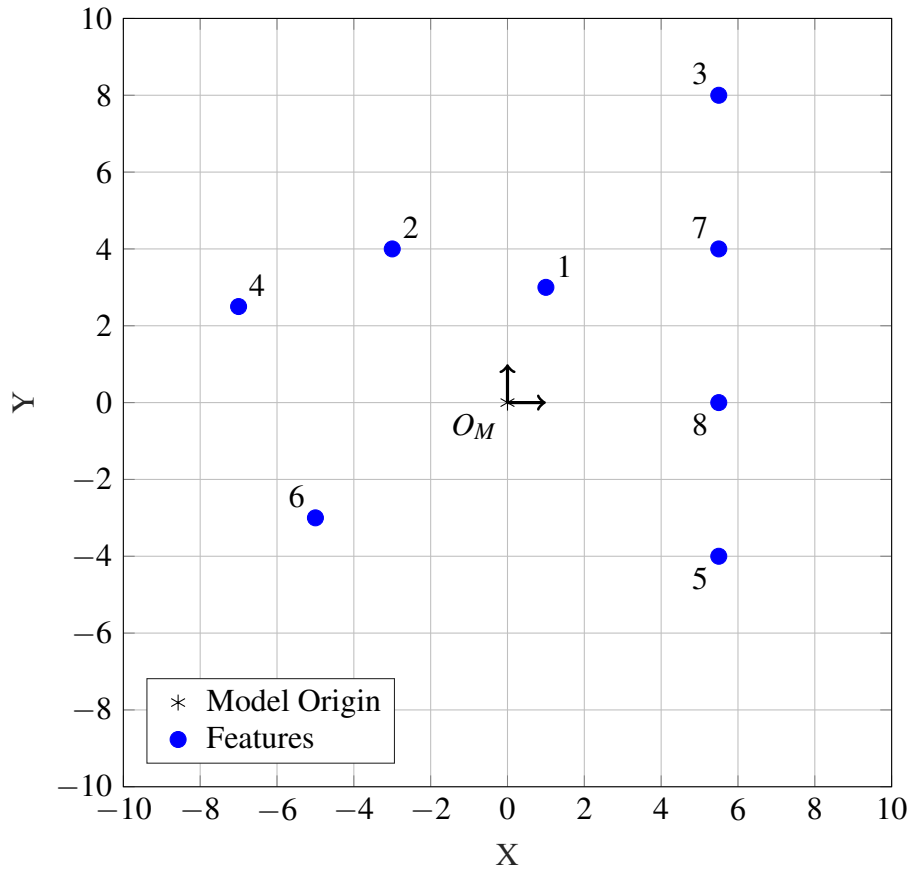


Figure 4.2: Model example depicting congruent base scenario

bases, then the two bases should not be checked for ambiguities. The congruency tolerance is usually determined as a function of the quantization parameter q_{pose} .

4.1.2.2 Algorithm: Screening

As mentioned in Chapter 3, the screening procedure can be broken down into 3 phases: identify similarity sets, collate the sets into to unique instances of ambiguous constellations, then determine relative transformations between each occurrence of the ambiguous constellations within the feature map. Again, these steps are collated into pseudo code in Algorithm 4.2 where the function "DETECT AMBIGUITIES" consumes a hash table H and layers database L and provides a database of ambiguities A as a result.

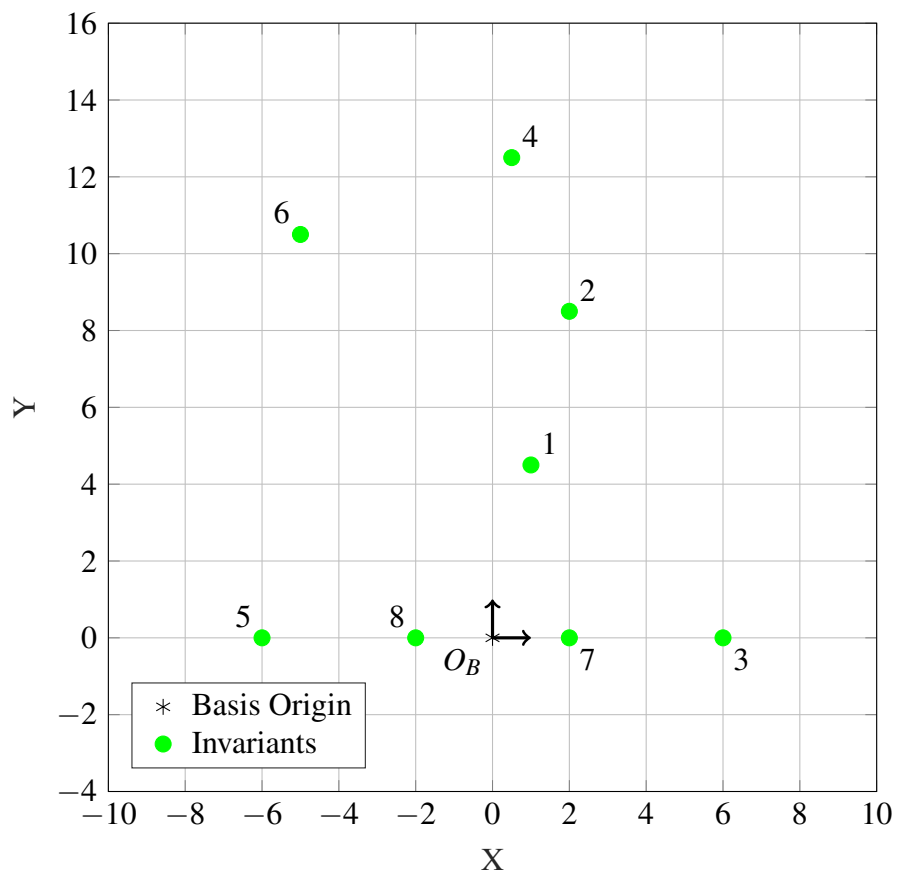


Figure 4.3: Congruent bases formed via two overlapping basis origin points

Algorithm 4.2 Screening Phase

```
1: procedure DETECTAMBIGUITIES( $H, L$ )  $\triangleright$  Create database of ambiguous constellations  $A$ 
2:   for  $i \leftarrow 1, \text{length}(H)$  do  $\triangleright$  Loop through all buckets
3:     if  $\text{length}(H[i][2]) > 1$  then  $\triangleright$  There must be more than one layer in the bucket
4:        $C \leftarrow \text{nchoose2}(\text{length}(H[i][2]))$ 
5:       for  $j \leftarrow 1, \text{length}(C)$  do  $\triangleright$  Loop through all layer combinations
6:          $[lhs, rhs] \leftarrow L[H[i][2][C[j][1]]], L[H[i][2][C[j][2]]]$ 
7:          $\delta \leftarrow \text{distance}(o_{lhs}, o_{rhs})$ 
8:         if  $\delta < t_{cong}$  then  $\triangleright$  Check for congruent bases
9:           continue
10:        end if
11:        for  $k \leftarrow 1, \text{length}(\vec{b}_{lhs}^q)$  do  $\triangleright$  Loop through keys in this layer
12:           $X \leftarrow \text{find}(\vec{b}^q[k], \vec{b}_{rhs}^q)$ 
13:        end for
14:        if  $\text{length}(X) > 0$  then
15:           $k \leftarrow \text{length}(X)$ 
16:          repeat
17:             $Z \leftarrow \text{nchoosek}(\text{length}(X), k)$ 
18:            for  $kk \leftarrow 1, \text{length}(Z)$  do  $\triangleright$  Loop through permutations
19:               $W \leftarrow \text{updateSimilarityInfo}(k, lhs, rhs, Z, X)$ 
20:            end for
21:             $k \leftarrow k - 1$ 
22:          until  $k < 1$ 
23:          end if
24:        end for
25:        for  $j \leftarrow 1, \text{length}(W)$  do  $\triangleright$  Loop through all similarity sets by vertex count
26:           $r_s \leftarrow \text{length}(W[1])$ 
27:          while  $r_s > 0$  do
28:             $A \leftarrow \text{addConstellation}(W[1][1], W[1][2])$ 
29:             $h_s \leftarrow \text{removeMatchingConstellations}(W)$ 
30:            for  $k \leftarrow 1, \text{length}(A_{layers})$  do  $\triangleright$  Loop through the number of occurrences
31:               $G \leftarrow \text{determineCentroids}(A_{mids}[k])$ 
32:               $H \leftarrow \text{centroidOrientation}(G, A_{mids}[k])$ 
33:               $A \leftarrow \text{addMetaInfo}(G, H)$ 
34:               $L[A_{layers}[k]] \leftarrow A_{id}[k]$ 
35:            end for
36:             $C_o \leftarrow \text{nchoose2}(A_{layers})$ 
37:            for  $k \leftarrow 1, \text{length}(C_o)$  do  $\triangleright$  Get transform info on each occurrence
38:               $A \leftarrow \text{addTransform}(A, k)$ 
39:            end for
40:             $r_s \leftarrow r_s - h_s$ 
41:          end while
42:        end for
43:      end if
44:    end for
45:    return  $A$ 
46: end procedure
```

4.2 Localization Pipeline

Before diving into the recognition phase, the logical flow of an example localization pipeline using Geometric Hashing to associate features should be shown to provide context. Figure 4.4 shows a possible localization pipeline using the state estimation method. Primitive features are first extracted from the scene using the latest laser measurements and fed into the hash function. Upon generating local bases, each basis is compared to bases stored in the hash table to find the best match at the feature recognition step. Once the most likely match is chosen, the invariants from the matched basis will correspond to model identifiers in the map. Lastly, resultant feature positions from the matching model identifiers are fed into a measurement model in order to update the vehicle states. Although it is not strictly necessary to use prior information at the feature extraction step as shown in Figure 4.4, knowledge of prior information can help reject instantaneous outliers should the incorrect association be chosen. On the other hand, prior information could also help propagate erroneous associations over time.

4.2.1 The Recognition Phase

With the completion of map generation, analysis and hash table construction, all the collected information can now be used during an online recognition sequence. As discussed in Chapter 2, the primary goal of the online recognition phase is to find the correct transformation, if it exists, between the scene features and the models accumulated in the training phase. This proper transformation can be found by determining correct associations of individual features and feature groups from the scene with those in the database. If no possible associations are determined, then the scene is likely not well represented in the database.

Since the recognition phase begins very similarly to the training phase, many of the same feature filtering techniques such as feature collision detection and basis limiting are applicable. Even though some of these techniques can reduce downstream computation, it is still important to evaluate the trade-offs of applying these techniques and meeting the real-time constraints of the system. Before diving into semantics with how the scene feature associations are determined, the role noise plays in the quality of the association step must first be addressed.

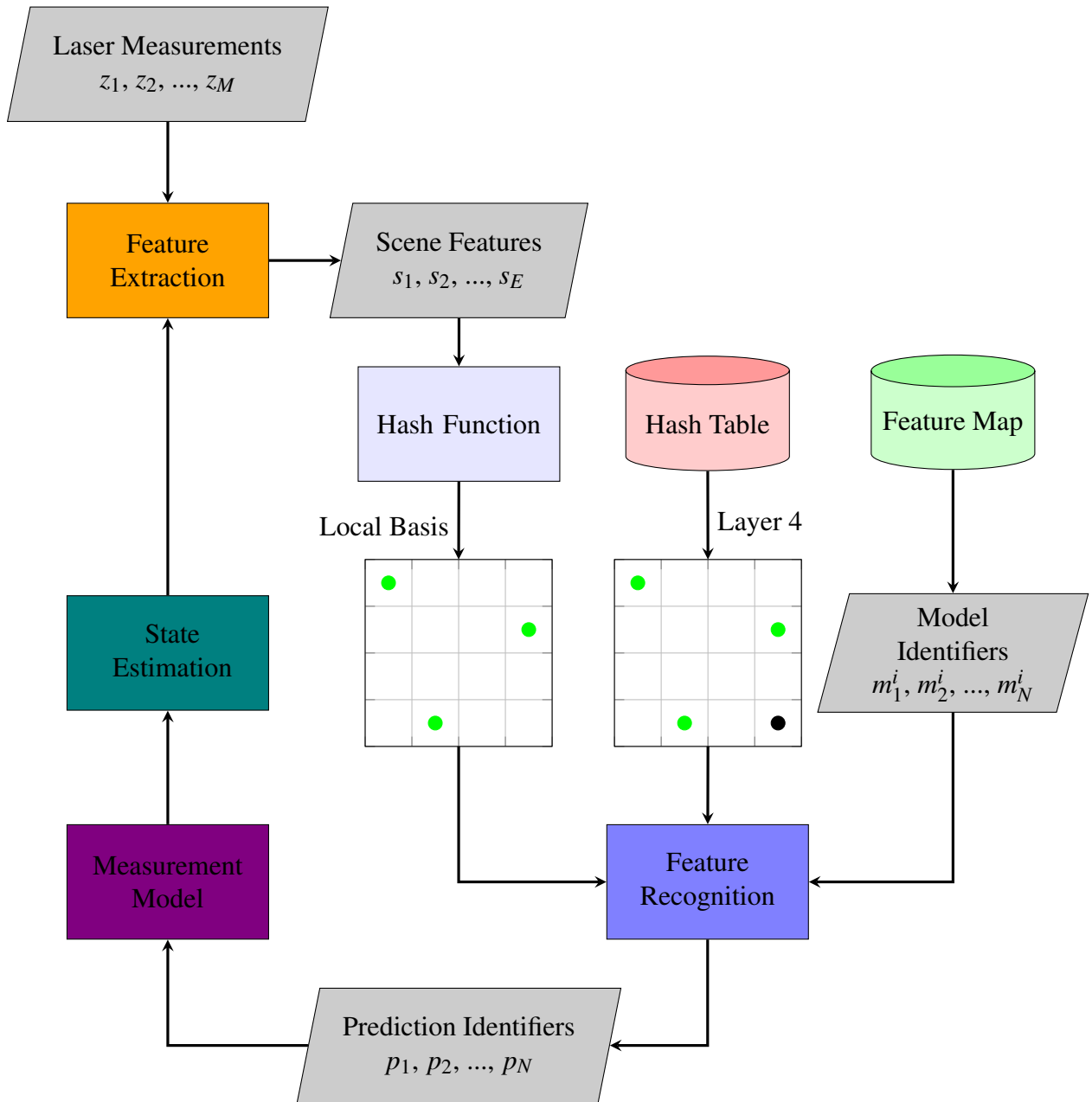


Figure 4.4: Geometric hashing localization flowloop

4.2.1.1 Noise Mitigation

If the primitive features extracted from each scene are noisy, obtaining a sufficient amount of features associations at each recognition step becomes difficult. The sensitivity of geometric hashing that results from noise was briefly discussed in Chapter 2 but no formulation or model was presented to mitigate the negative effects noise can introduce.

In the work from *Isadore Rigoutsos* a quantitative characterization of noise was conducted for invariants that were allowed to undergo similarity or affine transformations (*i.e.* rotation, translation, scaling, and shearing) [7]. Because each invariant is defined by a basis, and each basis by noisy feature positions, *Rigoutsos* discovered that the resultant uncertainty on the invariant is ultimately a function of both the type of transformation used as well as the proximity of the invariant to the basis origin.

Assuming that the noise on each extracted feature is Gaussian white noise, then the resulting noise on each invariant can also be approximated by a Gaussian distribution with the covariance matrix shown in Equation (4.8)

$$\Sigma_s = \frac{(4\|(\mu_s, \nu_s)\|^2 + 3)\sigma^2}{2\|\vec{p}_2 - \vec{p}_1\|^2} \quad (4.8)$$

In Equation (4.8), \vec{p}_1 and \vec{p}_2 are the extracted feature positions comprising the basis pair, μ_s and ν_s are the similarity transformation invariant positions, and σ is the variance on the extracted features [26].

Since the contributions of this thesis operate only under rigid transformation invariance as mentioned in Section 2.1, it is necessary to obtain a similar characterization under these circumstances. However, when applying the same approach to the rigid formulation, the resultant distribution was found to no longer be Gaussian. Because rigid transformations preserve scaling, the noisy invariants positions smear in an angular direction more noticeably than the radial. This results in a new "curved slot" shaped distribution that represents a smaller subset of the Gaussian distribution for the similarity transformation. As such, the new distribution can be over-bounded using Equation (4.8) as shown in Equation (4.9) [26].

$$\Sigma_s = \frac{(4\|(\mu_s, v_s)\|^2 + 3)\sigma^2}{2\|\vec{p}_2 - \vec{p}_1\|} \quad (4.9)$$

After obtaining information on the anticipated noise of the invariants, it is also important to consider how precision lost during the quantization step can also play a critical role during recognition. For example, if an invariant represented in continuous space is far away from its quantized position and holds a positional uncertainty magnitude close in value to q_{pose} , there exists a reasonable likelihood that the true position of the invariant is in a neighboring bin. To incorporate this, a weighting scheme can be applied to the resultant bin and all the neighbors such that the neighbors receive a lower weight than the center. Then, the neighboring bins can also be taken in to consideration as candidates for association.

4.2.2 Candidate Selection Methods

Chapter 2 provided a high overview as to how features from the local scene are associated to their mapped counterparts. However, there was no mention of how to remain robust to ambiguities and measurement uncertainty or noise. This section provides six different methods, Algorithms 4.3-4.8, to determine the best probable matches given noisy measurements and highly ambiguous geometries. The different methods can be split up in two different categories: feature-matching and basis-matching. In the feature-matching approaches, the best possible matches are determined on a feature-by-feature instance where certain matches may not be part of the same basis in the hash table. By contrast, in the basis-matching approaches, the best possible matches are selected from the basis that obtains the highest voting score.

Algorithm 4.3 is a basis-matching approach that provides a primitive association selection schema where the resultant features associations are simply taken from the basis layer that receives the most votes. Algorithm 4.4 is a feature-matching technique where the associations of each detection are obtained by identifying the most occurring match for each feature (*i.e.* the distributional mode). As each candidate layer is checked for associations, a tally is recorded for each unique feature identifier per detection. Then the feature identifier with the most tallies is deemed the association. Algorithm 4.5 is another feature-matching approach very similar

to Algorithm 4.4. Instead of searching through all candidate layers, only the candidates that recorded more matches than the upper-third percentile of matches for the set of candidates is searched. For example, if there are five candidate layers where three contain at least 7 matches and the other two each contain four matches, the latter two would be omitted from selection due to insufficient features when compared to the others. Algorithm 4.6 continues to build off of the approach in Algorithm 4.5. Similarly, Algorithm 4.6 is a feature-matching approach where candidates are first pruned by the upper-third percentile of matches. However, instead of taking the most occurring feature as was done in Algorithms 4.4-4.5, the weighting score is used based on the mahalanobis distance of the detection from the mapped feature. Algorithm 4.7 is a fusion between the approaches presented in Algorithms 4.5-4.6. While maintaining the feature-matching approach, associations are instead selected by averaging the weighting scores across all candidates and selecting the highest value. This approach provides the most flexibility to quantization loss and noise. Lastly, Algorithm 4.8 presents a basis-matching approach where the set of associations is determined by the highest sum of weight scores for each candidate layer.

Algorithm 4.3 Associate Features 1

<pre> 1: procedure SELECTASSOCIATION(Z) 2: $m \leftarrow 0, m_i \leftarrow 0$ 3: for $i \leftarrow 1, \text{length}(Z)$ do 4: if $Z[i][3] > m$ then 5: $m \leftarrow s$ 6: $m_i \leftarrow i$ 7: end if 8: end for 9: $P \leftarrow Z[m_i][1]$ 10: return P 11: end procedure </pre>	<pre> ▷ Select association based on weight score ▷ Loop through all candidate matches ▷ Cache the index with the most matches </pre>
---	--

Algorithm 4.4 Associate Features 2

```
1: procedure SELECTASSOCIATION( $Z$ )           ▷ Select the candidate from the mode of the
   distribution
2:    $m \leftarrow 0, m_i \leftarrow 0$ 
3:   for  $i \leftarrow 1, \text{length}(Z)$  do           ▷ Loop through all candidate matches
4:     if  $Z[i][3] > m$  then                       ▷ Cache the index with the most matches
5:        $m \leftarrow s$ 
6:        $m_i \leftarrow i$ 
7:     end if
8:   end for
9:   for  $i \leftarrow 1, \text{length}(Z)$  do           ▷ Loop through all candidate matches again
10:     $B_m \leftarrow \text{updateBallotList}(Z)$  ▷ Add/update entries in ballot for each feature: update
   the tally
11:   end for
12:   for  $i \leftarrow 1, \text{length}(B_m)$  do           ▷ Loop through list of ballots
13:      $w \leftarrow 0, w_i \leftarrow 0$ 
14:     for  $j \leftarrow 1, \text{length}(B_m[i])$  do       ▷ Loop through all features in ballot
15:        $q \leftarrow \frac{B_m[i][j][1]}{B_m[i][j][2]}$ 
16:       if  $q > w$  then                             ▷ Cache the highest weight
17:          $w_i \leftarrow B_m[i][j]$ 
18:          $w \leftarrow q$ 
19:       end if
20:     end for
21:      $P \leftarrow B_m[i][w_i]$ 
22:   end for
23:   return  $P$ 
24: end procedure
```

Algorithm 4.5 Associate Features 3

```
1: procedure SELECTASSOCIATION( $Z$ )    ▷ Select the upper third percentile of candidates,
   then use the distributional mode
2:    $m \leftarrow 0, m_i \leftarrow 0$ 
3:   for  $i \leftarrow 1, \text{length}(Z)$  do    ▷ Loop through all candidate matches
4:     if  $Z[i][3] > m$  then    ▷ Cache the index with the most matches
5:        $m \leftarrow s$ 
6:        $m_i \leftarrow i$ 
7:     end if
8:   end for
9:    $t \leftarrow \frac{2 * Z[m_i][3]}{3}$ 
10:  for  $i \leftarrow 1, \text{length}(Z)$  do    ▷ Loop through all candidate matches again
11:    if  $Z[m_i][3] \geq t$  then
12:       $B_m \leftarrow \text{updateBallotList}(Z)$     ▷ Add/update entries in ballot for each feature:
   update the tally
13:    end if
14:  end for
15:  for  $i \leftarrow 1, \text{length}(B_m)$  do    ▷ Loop through list of ballots
16:     $w \leftarrow 0, w_i \leftarrow 0$ 
17:    for  $j \leftarrow 1, \text{length}(B_m[i])$  do    ▷ Loop through all features in ballot
18:       $q \leftarrow \frac{B_m[i][j][1]}{B_m[i][j][2]}$ 
19:      if  $q > w$  then    ▷ Cache the highest weight
20:         $w_i \leftarrow B_m[i][j]$ 
21:         $w \leftarrow q$ 
22:      end if
23:    end for
24:     $P \leftarrow B_m[i][w_i]$ 
25:  end for
26:  return  $P$ 
27: end procedure
```

Algorithm 4.6 Associate Features 4

```
1: procedure SELECTASSOCIATION( $Z$ )    ▷ Select the upper third percentile of candidates,
   then use the highest weighting
2:    $m \leftarrow 0, m_i \leftarrow 0$ 
3:   for  $i \leftarrow 1, \text{length}(Z)$  do                                ▷ Loop through all candidate matches
4:     if  $Z[i][3] > m$  then                                          ▷ Cache the index with the most matches
5:        $m \leftarrow s$ 
6:        $m_i \leftarrow i$ 
7:     end if
8:   end for
9:    $t \leftarrow \frac{2 * Z[m_i][3]}{3}$ 
10:  for  $i \leftarrow 1, \text{length}(Z)$  do                                ▷ Loop through all candidate matches again
11:    if  $Z[m_i][3] \geq t$  then
12:       $B_m \leftarrow \text{updateBallotList}(Z)$  ▷ Add/update entries in ballot for each feature: use
   the max weight value
13:    end if
14:  end for
15:  for  $i \leftarrow 1, \text{length}(B_m)$  do                                ▷ Loop through list of ballots
16:     $w \leftarrow 0, w_i \leftarrow 0$ 
17:    for  $j \leftarrow 1, \text{length}(B_m[i])$  do                            ▷ Loop through all features in ballot
18:       $q \leftarrow \frac{B_m[i][j][1]}{B_m[i][j][2]}$ 
19:      if  $q > w$  then                                              ▷ Cache the highest weight
20:         $w_i \leftarrow B_m[i][j]$ 
21:         $w \leftarrow q$ 
22:      end if
23:    end for
24:     $P \leftarrow B_m[i][w_i]$ 
25:  end for
26:  return  $P$ 
27: end procedure
```

Algorithm 4.7 Associate Features 5

```
1: procedure SELECTASSOCIATION( $Z$ )    ▷ Select the upper third percentile of candidates,
   then find the average between mode and mahalanobis distance
2:    $m \leftarrow 0, m_i \leftarrow 0$ 
3:   for  $i \leftarrow 1, \text{length}(Z)$  do                                ▷ Loop through all candidate matches
4:     if  $Z[i][3] > m$  then                                          ▷ Cache the index with the most matches
5:        $m \leftarrow s$ 
6:        $m_i \leftarrow i$ 
7:     end if
8:   end for
9:    $t \leftarrow \frac{2 * Z[m_i][3]}{3}$ 
10:  for  $i \leftarrow 1, \text{length}(Z)$  do                                ▷ Loop through all candidate matches again
11:    if  $Z[m_i][3] \geq t$  then
12:       $B_m \leftarrow \text{updateBallotList}(Z)$     ▷ Add/update entries in ballot for each feature:
   sum weights, increase tally
13:    end if
14:  end for
15:  for  $i \leftarrow 1, \text{length}(B_m)$  do                                ▷ Loop through list of ballots
16:     $w \leftarrow 0, w_i \leftarrow 0$ 
17:    for  $j \leftarrow 1, \text{length}(B_m[i])$  do                            ▷ Loop through all features in ballot
18:       $q \leftarrow \frac{B_m[i][j][1]}{B_m[i][j][2]}$ 
19:      if  $q > w$  then                                              ▷ Cache the highest weight
20:         $w_i \leftarrow B_m[i][j]$ 
21:         $w \leftarrow q$ 
22:      end if
23:    end for
24:     $P \leftarrow B_m[i][w_i]$ 
25:  end for
26:  return  $P$ 
27: end procedure
```

Algorithm 4.8 Associate Features 6

```
1: procedure SELECTASSOCIATION( $Z$ )    ▷ Select association based on weight score
2:    $w \leftarrow 0, w_i \leftarrow 0$ 
3:   for  $i \leftarrow 1, \text{length}(Z)$  do                                ▷ Loop through all candidate matches
4:      $s \leftarrow \text{sum}(Z[i][2])$                                        ▷ Sum the weights
5:     if  $s > w$  then                                              ▷ Cache the index of highest weight
6:        $w \leftarrow s$ 
7:        $w_i \leftarrow i$ 
8:     end if
9:   end for
10:   $P \leftarrow Z[w_i][1]$ 
11:  return  $P$ 
12: end procedure
```

4.2.2.1 Algorithm: Recognition

A noteworthy feature of the recognition algorithm given in Algorithm 4.9 is that certain steps can execute concurrently. This trait can introduce large performance gains particularly when creating local bases and tallying votes for candidates. If the function "RECOGNIZE FEATURES" were to run in a concurrent processing framework, each basis pair could be checked for matching feature identifiers independently. In other words, the for loop in step 4 could be passed out to a job pool.

4.2.3 Verifying the Association

Since the implementation of Geometric Hashing in this thesis is intended for use in highly automated driving maneuvers in urban environments, safety is a critical concern. In order to address this, the last procedural step conducted at each recognition step is to apply some form of verification that the localization solution update will not introduce Hazardous Misleading Information (HMI) to the system. Part of this approach was discussed in Section 3.3 where computing all instances of ambiguous constellations can provide deterministic upper bounds on possible error which could allow for greater understanding of integrity risk, but this does not reduce the likelihood of the risk each update.

For this implementation of Geometric Hashing Localization, the previously mentioned verification is augmented with two additional checks. Even though the algorithm does not need to rely on prior updates to function properly, the first check introduces this as a weak dependency by setting a sanity check threshold on max allowable traversal distance since the last update. This allows the algorithm to throw away updates that attempt to re-position the vehicle large distances from the previous position. The second check is only valid when using feature-matching methods. Since the basis-matching approach automatically ensures that all associated features are within a reasonable proximity to one another, a safety check must be applied for the feature-matching case. If any associated feature violates a certain proximity threshold, then it is likely that HMI is present and the association is discarded. For simplicity, the threshold can easily be determined as a function of the vehicle sensory Field of View (FoV).

Algorithm 4.9 Recognition Phase

```
1: procedure RECOGNIZEFEATURES( $H, L, s_i, s_x, s_y$ )  $\triangleright$  Associate scene features  $S$  with those
   in the map  $M$ 
2:    $l \leftarrow 0$ 
3:    $C \leftarrow nchoose2(length(\vec{s}_i))$   $\triangleright$  Returns a  $c_s \times 2$  matrix of combinations
4:   for  $i \leftarrow 1, length(C)$  do  $\triangleright$  Loop through combinations
5:      $\vec{c} \leftarrow (C[i][1], C[i][2])$ 
6:      $\delta \leftarrow distance(\vec{s}_x[c_x], \vec{s}_y[c_y])$   $\triangleright$  Get the scalar distance
7:      $\theta \leftarrow orientation(\vec{s}_x[c_x], \vec{s}_y[c_y])$   $\triangleright$  Get the angle of basis w.r.t. scene frame
8:     if  $\delta < b_{limit}$  then  $\triangleright$  Ensure basis pair are within distance threshold
9:        $l \leftarrow l + 1$ 
10:       $\vec{o}_l \leftarrow midpoint(\vec{s}_x[c_x], \vec{s}_y[c_y])$   $\triangleright$  Average the position to get basis origin
11:       $R \leftarrow rotmat(\theta)$ 
12:      for  $j \leftarrow 1, length(\vec{s}_i)$  do  $\triangleright$  Loop through all scene features
13:         $\vec{p} \leftarrow (\vec{s}_x[j], \vec{s}_y[j])$ 
14:         $\vec{t} \leftarrow \vec{p} - \vec{o}_l$ 
15:        if  $r_{incl} < distance(t_x, t_y)$  then  $\triangleright$  Ensure point is within inclusion radius
16:           $\vec{b} \leftarrow transform(\vec{t}, R, s_{trans}/\delta)$   $\triangleright$  Transform point into basis frame
17:           $H_s \leftarrow hash(\vec{b}, q_{pose}, l)$   $\triangleright$  Quantize and hash the transformed point
18:           $L_s \leftarrow updateLayerInfo(j, \vec{b})$   $\triangleright$  Store all candidate layers
19:           $L_s \leftarrow neighborhood(H_c)$   $\triangleright$  Store neighboring bins to account for noise
20:        end if
21:      end for
22:       $B \leftarrow createBallot(L_s)$   $\triangleright$  Key-value pairs of local layers and occurrences
23:       $[e, t] \leftarrow maxCount(B)$   $\triangleright$  Find the entry with the most counts
24:      if  $t > 2$  then
25:         $a \leftarrow round(\frac{2t}{3})$   $\triangleright$  Capture upper 3rd percentile of max occurrences
26:        for  $j \leftarrow 1, length(B)$  do  $\triangleright$  Loop through ballot
27:          if  $B[j][2] \geq a$  and  $B[j][2] > 2$  then
28:             $L_c \leftarrow B[j][1]$ 
29:          end if
30:        end for
31:      else
32:        continue
33:      end if
34:      for  $j \leftarrow 1, length(L_c)$  do  $\triangleright$  Loop through candidate layers
35:        for  $k \leftarrow 1, length(L_c[j])$  do  $\triangleright$  Loop through points in layer
36:           $[z_{idx}, m_{min}] \leftarrow minMahalanobisDist(L_c[j][k], L[j])$ 
37:          if  $m_{min} > 3$  then  $\triangleright$  Threshold results
38:             $Z \leftarrow addMatch(L[j][z_{idx}], \frac{1}{m_{min}})$   $\triangleright$  Add match and weight
39:          end if
40:        end for
41:      end for
42:    end if
43:  end for
44:  return  $selectAssociation(Z)$ 
45: end procedure
```

Now with the full suite of algorithms presented in this chapter, experiments can be conducted to vilify the theory.

Chapter 5

Simulation and Experimental Test Design

The first measure taken to verify that Geometric Hashing Localization is a suitable localization method was to isolate the feature association step from the other aspects of the full localization update. To accomplish this, a software simulation was developed utilizing generated wheel speed and steering angle data as an input to simulate driving the system along a path through a pre-built map. Using the known FOV information from the experimental vehicle, the simulation can dispense extracted features to the association step without needing to run the full extraction pipeline. To quantify the results of the data associations produced in the simulation, the resultant measurement updates can be compared to a reference localization solution.

Following this procedure, in order to further verify the solution is capable of running in highly time-critical applications, an implementation of Geometric Hashing Localization written in C++ was integrated into the existing framework presented in *Kümmerle et al* [23]. Here the feature association module was replaced with the geometric hashing pipeline introduced in Chapter 4.

In addition to describing the two frameworks mentioned above, this chapter will also provide a detailed description of the test environments, the robotic vehicle platform used to collect data, and the reference solution used to verify the results. Finally, the chapter will close with several goals and expected outcomes for the experimentation.

5.1 Hardware Setup

In order to collect real world data for use during the experimentation presented in this thesis, a vehicle platform with a sufficient sensor suite is needed. To fulfill these needs, a Mercedes-Benz E-Class, with the given name "BerthaOne", as depicted in Figure 5.1 was chosen. BerthaOne is a fully robotic vehicle capable of performing highly automated cooperative



Figure 5.1: BerthaOne robotic vehicle depicting computing hardware configuration [1].

driving maneuvers and has a long list of accolades from various international robotics challenges [1]. As shown in Figure 5.2, the vehicle is equipped with a variety of monocular and stereo gray-scale and color cameras, multi-range radars, and a single long range lidar to provide sufficient sensory redundancy to the automation software.

BerthaOne contains two on-board computing platforms with designated operational domains. The main autonomy computer consists of a Linux-based server running two Intel Xeon E5-2640v3 processors each containing 16 cores at 2.6 GHz clock frequencies, 64 GB of RAM, and a Nvidia GeForce GTX 980 Ti GPU. The motion execution computer consists of a real-time operating system and is tasked with providing low level control and command to the vehicle over the CAN bus [1].

For the on-board cameras, five BlackFly PGE-50S5M-C monocular cameras are used surrounding the vehicle. The two side mounted cameras use Lensagon BM2920S118 fisheye lenses to provide a wide-angle field of view (FoV), while the other three units use Lensagon BM4018S118 lenses. The two cameras behind the front windshield are used in a stereo configuration to provide depth information to the perception system. The color camera used for colored object detection mounted in the front of the vehicle is a BlackFly PGE-50S5C-C using the same Lensagon BM4018S118 lens. The long range lidar is an Ibeo LUX4 which provides 4 layers of ranging information. In addition to perceptive sensors, BerthaOne is also equipped with an OxtS TR3000 high precision GNSS/INS odometry solution to fuse wheel speed and steering angle sensors as well as a Ublox C94-M8P dual GNSS receiver for redundancy [1].

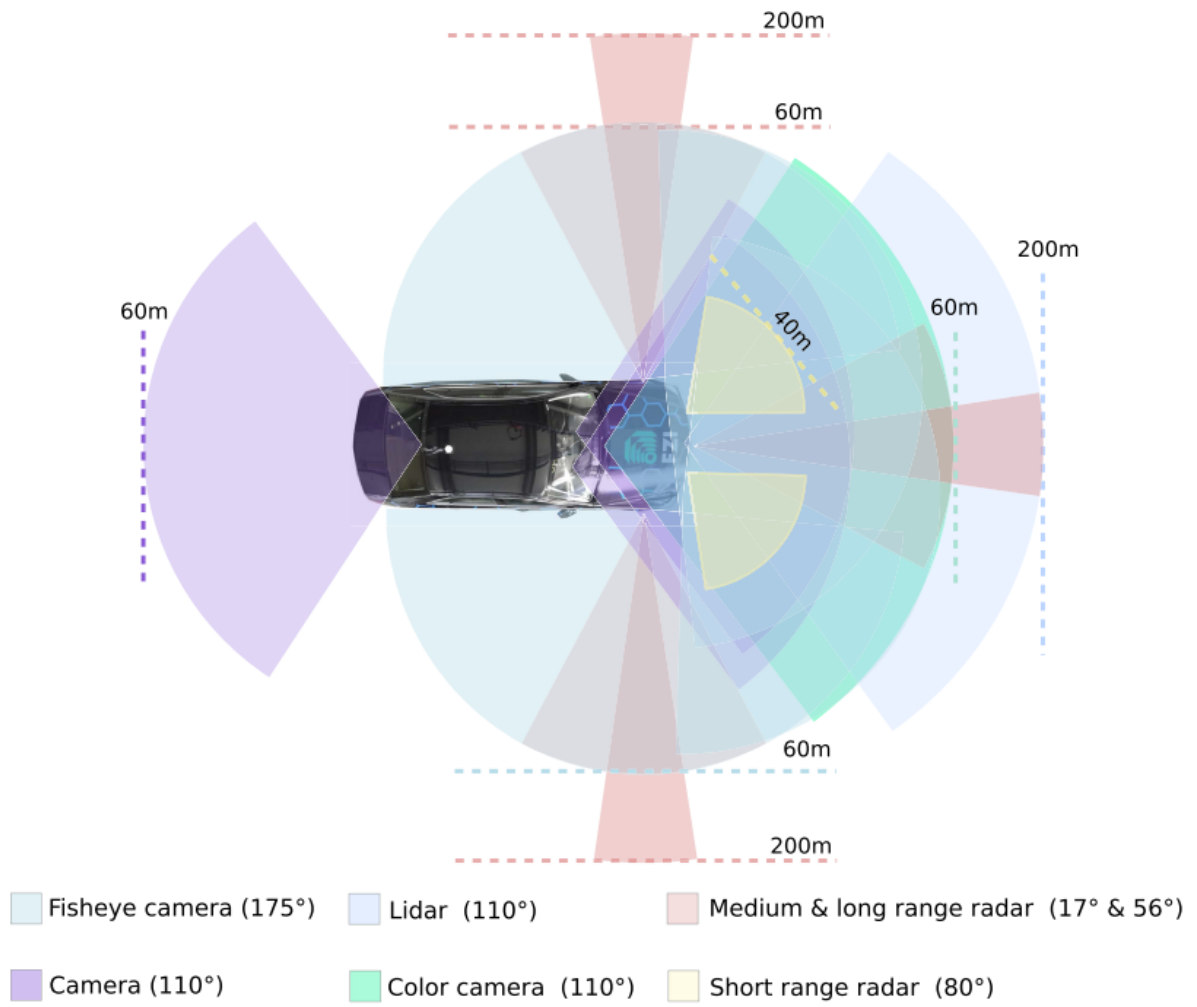


Figure 5.2: BerthaOne sensory field of view and layout [1].

Since the experimentation presented in this thesis is a continuation of the work presented in *Kümmerle et al* [23], the hardware and sensory additions to the vehicle since the work of *Tas et al* [1] remain in place. Notably, this boils down to the addition of four Velodyne VLP16 lidars with full 360 degree FoV mounted flat on the roof of the vehicle. Since each lidar is within direct line-of-sight and close proximity with one another, the internally rotating lasers are time synchronized and phase-locked at different offset so that no laser is ever projected in the direction of another sensor’s detector. Without this, the sensors are prone to reporting spurious false detections.

To run the software simulation portion of the experimentation, a desktop computer running Ubuntu Linux Xenial Xerus (16.04) was used. The system consisted of a single Intel i7-3700K processor containing 8 cores running at 3.5GHz, 16 GB of RAM, and an Nvidia GeForce GTX 660 GPU.

5.2 The Reference Solution

In order to effectively compare and evaluate Geometric Hashing Localization as a feasible method for autonomous driving precision and real-time performance, a pre-existing high-fidelity localization solution was used as a reference. The solution, first mentioned in Chapter 3 called *Surround View* [24], is a full six degrees-of-freedom (DoF) position update using a suite of cameras spanning all around the vehicle. Similar to the Geometric Hashing approach, *Surround View* also requires an *a priori* map created from images collected on a previous drive through an area.

Maps are generated offline by tracking keypoints throughout a sequence of images and then determining optimal landmark positions and the traveled vehicle path through a bundle adjustment method. Landmarks for the map are stored alongside the observed image keypoints in a planar grid structure known as the feature grid. When conducting a subsequent pass through the mapped environment, localization is achieved by providing a set of observations and landmarks from the feature grid based on the current predicted egopose of the vehicle and

associating them with current observations of the scene. After obtaining an initial GNSS position and heading, no additional sensors beyond the cameras are required to maintain continuous solution updates.

Surround View is also optimized to continuously improve the mapping quality by driving multiple passes through the desired localization region [27]. This helps increase the longevity of the solution as the physical environment changes over time because maps can be refined and updated as new features are introduced to the previous set of images.

Even though the path of vehicle during the collection drive is a component of the map, *Surround View* can still provide high-fidelity localization updates independent of the direction driven on subsequent passes [24]. When driving within proximity to the vehicle path from the mapping drive, *Surround View* is capable of providing positioning uncertainty under 1 centimeter as well as a rotation uncertainty under 0.2 degrees. These uncertainty values increase drastically as the vehicle moves away from the mapped vehicle path due to a decreased likelihood of matching keypoints between the map and scene. Given the sensor setup described in Section 5.1, the BerthaOne platform has a sufficient camera rig to generate a concrete reference solution required to evaluate Geometric Hashing Localization.

5.3 Mapped Circuits

Data collections using the aforementioned platform were conducted on two urban circuits within the Baden-Württemberg region in Germany. In this section, each circuit will be introduced as well as the maps generated following the techniques described in Section 3.1.1 for these areas. The colored routes indicating the path of travel for the experimental vehicle were collected using the reference solution.

5.3.1 Sindelfingen Route

The first circuit, henceforth known as the "Sindelfingen Route", is a rounded trapezoidal-shaped circuit through a medium-density urban environment located in Sindelfingen, Germany. The circuit consists of two sharp left turns, one partial left turn, and one roundabout. The mapped portion of the circuit generated 738 pole-like features over the course of a 10.9 km



Figure 5.3: Driven route in Sindelfingen, Germany

drive. During the course of travel, the vehicle performed 1.75 laps of the circuit traveling in the counterclockwise direction. Figure 5.3 shows the course of travel as well as the starting and end points of the collection event. Figures 5.4-5.5 depict the positions of the pole-like features and traversal speed over the mapped area respectively.

5.3.2 Karlsruhe Route

The second circuit, known as the "Karlsruhe Route", is a course through a medium-density urban environment located in Karlsruhe, Germany. The circuit consists of several more complex lane markings over the Sindelfingen route. In addition, the course contains two roundabouts, one sharp right turn, two sharp left turns, and several lane changes. The mapped portion of the circuit generated 891 pole-like features over the course of a 12.4 km drive. The vehicle performed just over one full lap of the route traveling in the counterclockwise direction. Figure 5.6 shows the course of travel and Figures 5.7-5.8 show the pole-like feature positions and traversal speed over the mapped area respectively .

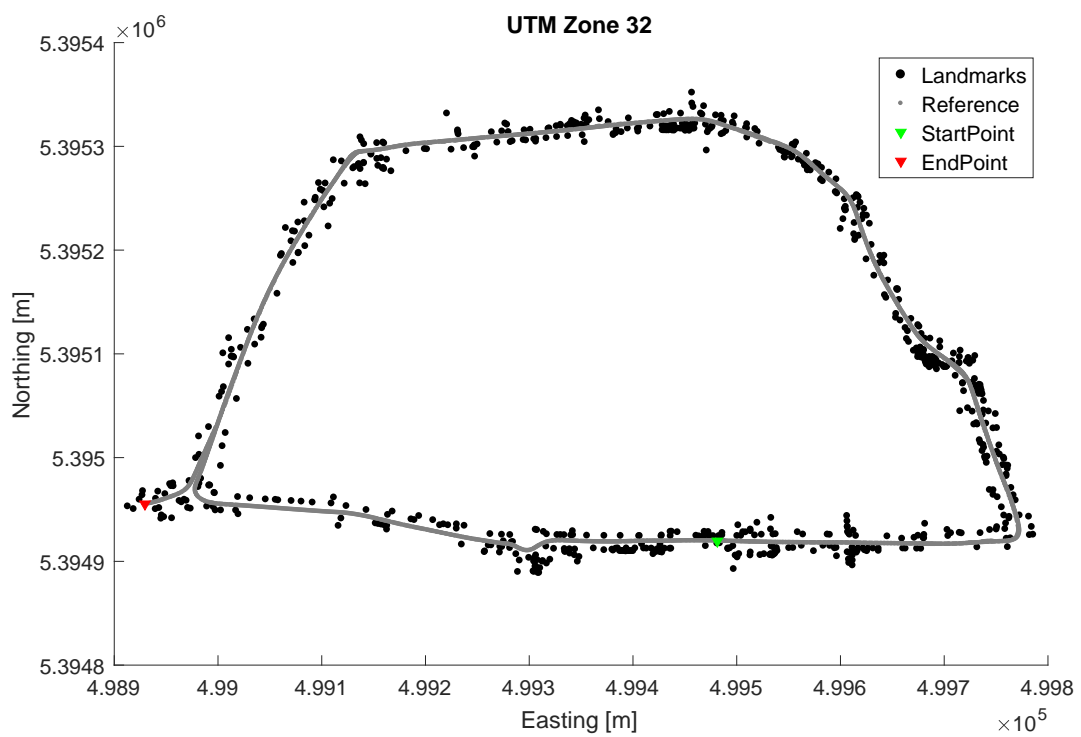


Figure 5.4: Extracted cylinders from Sindelfingen route

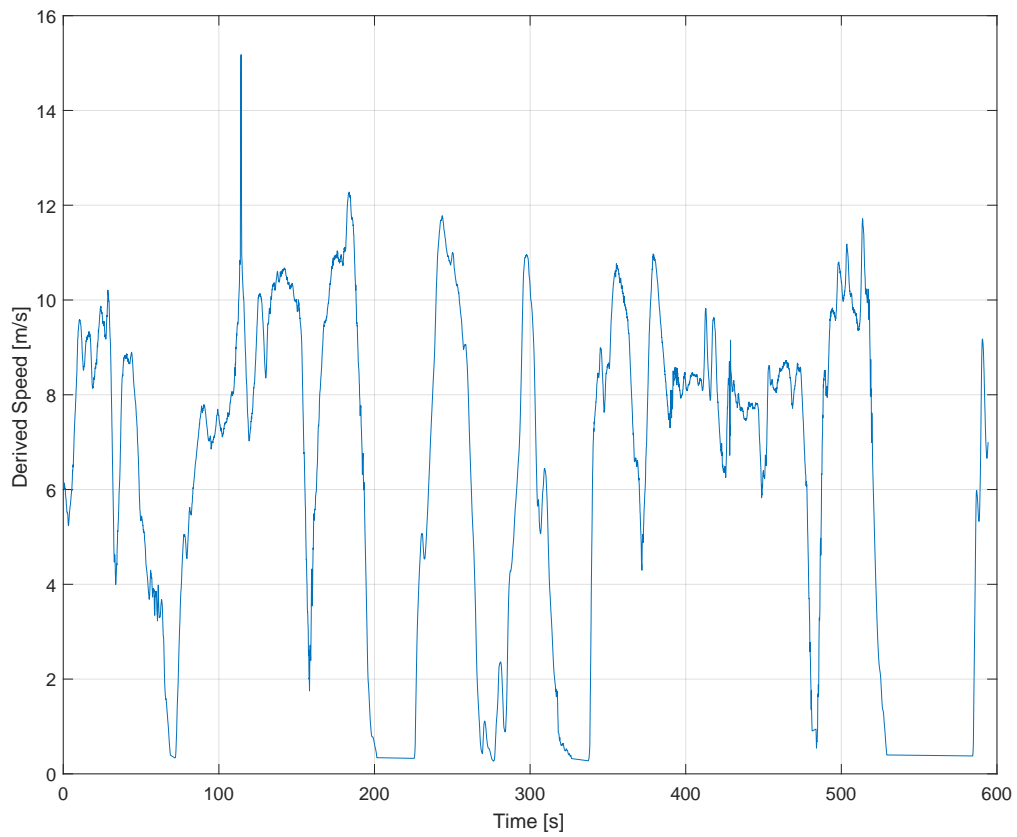


Figure 5.5: Derived speed of vehicle with smoothing



Figure 5.6: Driven route in Karlsruhe, Germany

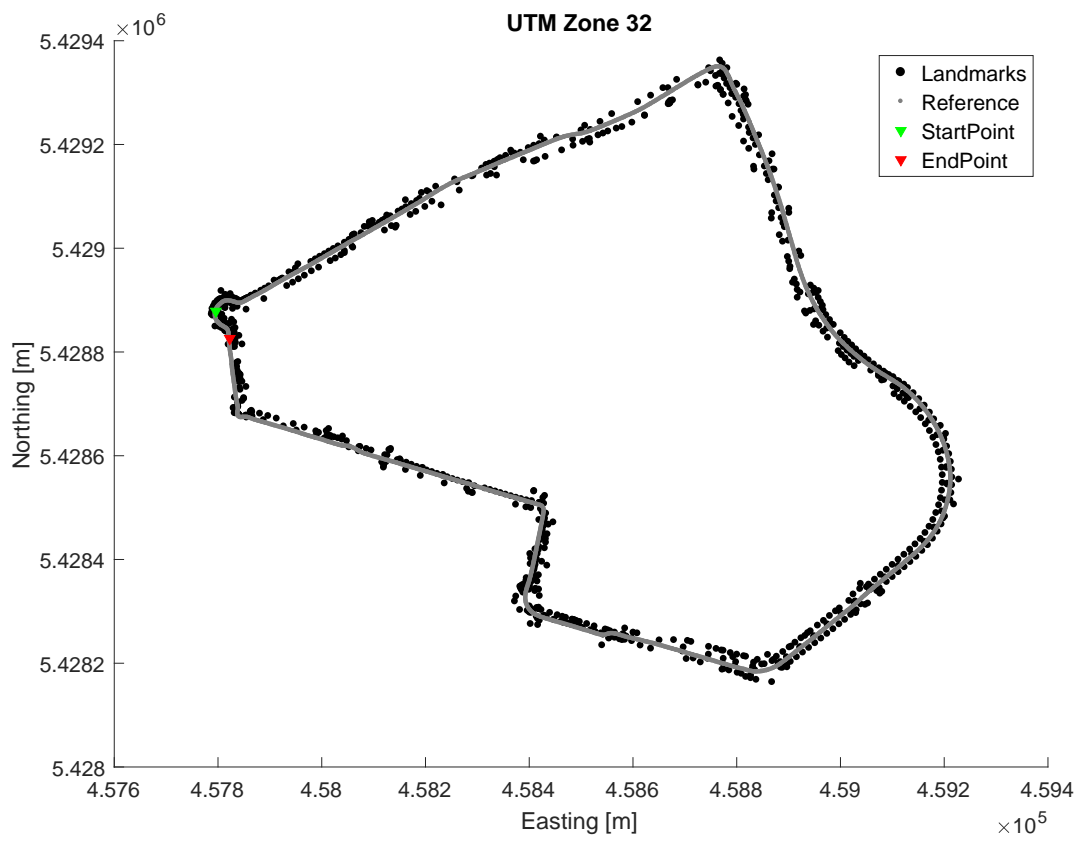


Figure 5.7: Extracted cylinders from Karlsruhe route

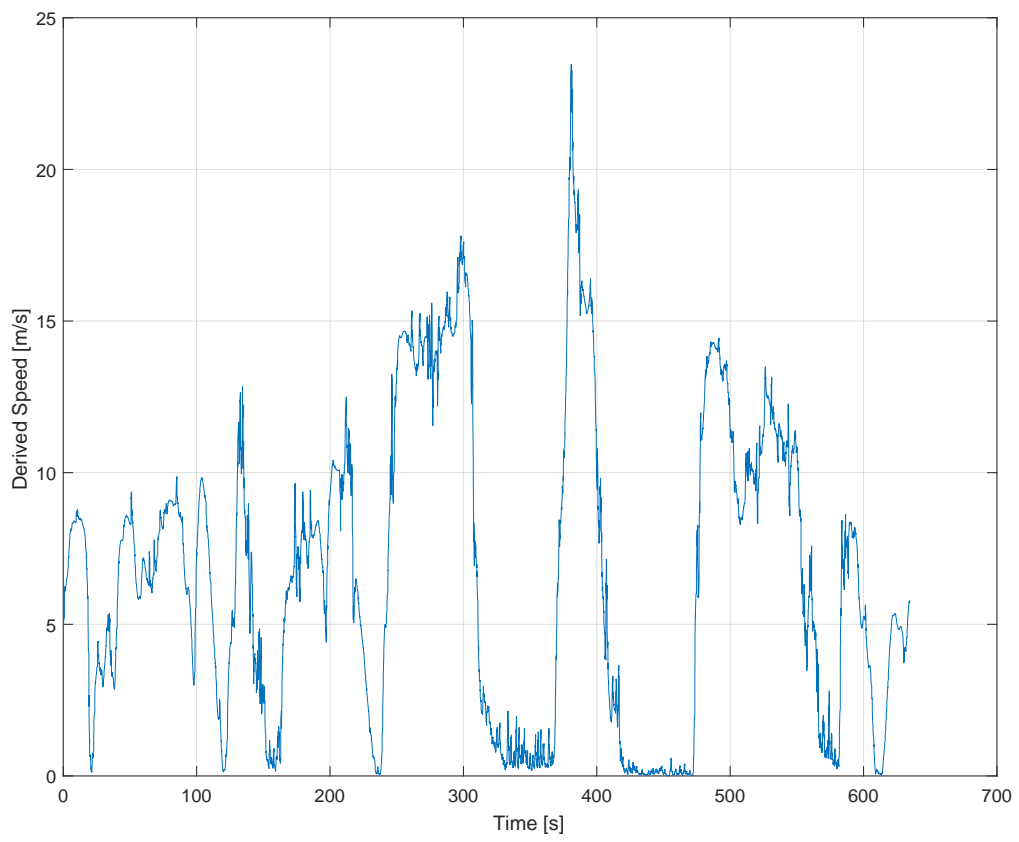


Figure 5.8: Derived speed of vehicle with smoothing

5.4 Simulation Framework

As an initial measure to understand how Geometric Hashing Localization performs as a viable localization solution, a mock scenario was generated and run through a simulation to test for accuracy, availability, and robustness to poor map geometry. Because the simulation was designed to directly evaluate the recognition phase of Geometric Hashing Localization, other aspects of the full localization pipeline, such as feature extraction, were severely generalized. For example, rather than simulating a full feature extraction process by working with raw laser data, feature poses and descriptors in the simulation are directly fed to the data association step based on the true simulated position of the vehicle and the given sensor FoV.

To setup the simulation, features can either be generated randomly along an arbitrary path or fed into the simulation from the maps of the two circuits presented above. In the case that features are generated, first an arbitrary combination of lines and curves are generated to simulate a vehicle trajectory. Then features are placed at random along various regions of the trajectory such that at least one feature is always visible anywhere along the path given an input sensor FoV. In order to simulate vehicle motion, wheel speed, steering angle, and accelerometer data were generated according to the kinematics of a standard bicycle model and an input acceleration profile along the trajectory. For non-generated data, these signal streams are already available in collected data from the BerthaOne vehicle. Lastly, in order to generalize the feature extraction process, measurements are generated as the vehicle moves along the true trajectory. At each sampled position, the range and bearing measurements from the vehicle to each feature in the FoV are then corrupted by Gaussian White Noise to emulate small errors in a realistic feature extraction procedure.

Following the localization flowchart presented in Figure 4.4 from Chapter 4, an Extended Kalman filter was developed using a 6 DoF dynamic bicycle model to predict position updates along the trajectory and fuse in feature association measurements as corrections. The equations of motion that makeup the model are defined in Equation (5.1) as presented in *Pepy et al* [28].

$$\begin{bmatrix} m(\ddot{x} - v_y \dot{\psi}) \\ m(\ddot{y} + v_x \dot{\psi}) \\ I_{zz} \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{C_f v_y \sin \delta}{v_x} + \frac{C_f L_f \dot{\psi} \sin \delta}{v_x} - C_f \delta \sin \delta \\ \frac{C_r L_r \dot{\psi}}{v_x} - \frac{C_f v_y \cos \delta}{v_x} - \frac{C_f L_f \dot{\psi} \cos \delta}{v_x} - \frac{C_r v_y}{v_x} + C_f \delta \cos \delta \\ \frac{C_r L_r^2 \dot{\psi}}{v_x} - \frac{C_f L_f v_y \cos \delta}{v_x} - \frac{C_f L_f^2 \dot{\psi} \cos \delta}{v_x} - \frac{C_r L_r v_y}{v_x} + C_f L_f \delta \cos \delta \end{bmatrix} \quad (5.1)$$

The input parameters in Equation (5.1) are as follows; m is the mass at the vehicle center, C_f and C_r are the front a rear cornering stiffness coefficients respectively, L_f and L_r are the lever arms from vehicle center to tire centers respectively, I_{zz} is the z-axis moment of inertia about the vehicle center and δ is the steering angle. The model assumes a linear tire model, all aerodynamic drag forces and suspension dynamics are negligible, and slip angles will always remain small. The remaining variables used to provide the tracked system states are shown in Equation (5.2).

$$X_s = \left[X \quad Y \quad \psi \quad v_x \quad v_y \quad \dot{\psi} \right]^T \quad (5.2)$$

From Equation (5.2), X and Y are the reference frame horizontal positions with positive X pointing east and positive Y pointing north, ψ is heading with zero pointing east and counterclockwise positive, v_x and v_y are the horizontal vehicle body velocities where positive v_x points out of vehicle front and positive v_y points out of vehicle left, and $\dot{\psi}$ is the counterclockwise positive vehicle yaw rate.

The time propagation interval for predicting new vehicle state estimates is explicitly chosen to be misaligned from the measurement sampling rate in order to stress the simulation with asynchronous prediction and correction updates. At each time propagation (*i.e.* integration step), the vehicle states are stepped forward in time using a 4th-order Runge-Kutta integration by using the vehicle wheel speed and steering angle as inputs to the system. Incoming measurements arriving within each propagation window are fed through the recognition phase to obtain the correct feature positions from the map. These positions are then evaluated in the feature observation model using the 2D range-bearing formulation as shown in Equation (5.3).

$$G_s = \begin{bmatrix} \sqrt{(p_x - X)^2 + (p_y - Y)^2} \\ \text{atan2}((p_y - Y), (p_x - X)) - \psi \end{bmatrix} \quad (5.3)$$

In Equation (5.3), p_x and p_y are the associated feature poses from the map in the absolute coordinate frame described above. Careful consideration must be given to ensure the resultant value for bearing stays within the bounds of $[-\pi, \pi]$.

In typical real-world scenarios, new measurements will only produce valid corrections for the state variables at the time they arrive. Because this mismatch of timestamps will be explicitly present in the simulation, an assumption is made that the integration time step is sufficiently small to account for any errors produced by correcting the state at a future time from when the measurement arrived. In order to more robustly solve this problem for measurements that are very delayed, a delayed-state filter could be used to track two copies of the state variables over time; one with the latest prediction, and the other from measurement signal with the lowest update frequency.

5.5 Localization Framework

In addition to verifying basic localization requirements of the algorithm using the simulation discussed in the previous section, it was also desired to implement Geometric Hashing Localization into an existing real-time framework for analysis. As mentioned earlier in this chapter, Geometric Hashing Localization was integrated into the existing pipeline from the work presented in *Kümmerle et al*[23] replacing the data association step with the recognition phase presented in this thesis.

Before proceeding with online localization, the map generated *a priori* is first run through the training and screening phases to generate the appropriate databases needed for recognition. Afterwards, the new framework is largely similar to the flowchart depicted in Figure 4.4. Each update cycle begins by conducting the cylindrical feature extraction process as described in Chapter 3 on a 3D point cloud representing a single revolution worth of laser measurements. If a sufficient amount of features is successfully extracted, the extracted feature positions and any

additional descriptors are handed off to the recognition phase of Geometric Hashing Localization. If recognition is successful, the corresponding map feature positions are provided back to the localization framework. Optionally, an estimate of the system's prior state can be provided to the recognition step to serve as a sanity check for the resultant mapped feature positions.

The major difference in this approach from the flowchart presented in Figure 4.4 is in the Measurement Model update step. The simulation approach introduced in Section 5.4 follows the flowchart very closely. In the real-time framework however, rather than running the mapped feature positions through a dedicated measurement model, the feature positions obtained after the recognition step are instead fed directly into a non-linear optimization framework. The framework employs a pose graph adjuster, which can adjust a window full of poses based on optimizing new poses and their respective confidence values with those in the window. The optimization step utilizes the Levenberg-Marquardt non-linear regression algorithm to determine the adjustment for the window [23]. Now that both experimental frameworks have been introduced, a discussion of results can begin in the next chapter.

Chapter 6

Experimental Results

In this chapter, an analysis of the results collected from the experimentation described in Chapter 5 are reported. The results are presented according to the data set used to acquire them. Each set of results includes two components: an offline collection of parameters, data structures, and databases, and the online results from the localization loop. First a set of result figures will be shown for each data set, followed with a brief discussion covering all the data scenarios for the simulation and the integrated localization solution respectively.

6.1 Simulation

This section covers the experimental results produced from the software simulation described in Section 5.4. Although a large variety of simulated scenarios were tested over the duration of this research, for reasons of brevity, only the most complex scenario was chosen to be presented in this thesis. The scenario involves a vehicle driving through an "S" shaped path. Starting from rest, the vehicle will accelerate linearly until reaching a maximum velocity while maneuvering through the curvature of the path, then decelerate linearly back to rest. This path and velocity profile combination was chosen to highlight the Geometric Hashing algorithm's ability to produce valid positioning corrections despite the non-linearities in both longitudinal and lateral motion cases.

The simulated vehicle parameters used to predict motion are shown in Table 6.1 along with the simulated LiDAR parameters in Table 6.2. The Kalman filter uses an integration time interval of 0.1 seconds and other filter modeling parameters can be seen in Table 6.3. Using this configuration, two data sets are reported representing a case where the map consists only of unique feature geometry and a case where ambiguities are present. Lastly, Algorithm ?? was used to determine matches during the recognition step for both data sets.

Table 6.1: Simulated Vehicle Parameters

Name	Value
Vehicle Mass [kg]	1500
Vehicle Length [m]	3
Vehicle Width [m]	1.5
Z-axis MOI [$kg \cdot m^2$]	1406.25
CG to Front Axle [m]	1.3
CG to Rear Axle [m]	1.3
Front Tire Cornering Stiffness [$\frac{N}{rad}$]	0.2
Rear Tire Cornering Stiffness [$\frac{N}{rad}$]	0.2

Table 6.2: Simulated LiDAR Parameters

Name	Min Value	Max Value
Range [m]	0.5	40
Azimuth [rad]	$-\pi$	π

Table 6.3: Kalman Filter Parameters

Name	Value
Horizontal Pos Variance [m^2]	0.25
Orientation Variance [rad^2]	0.25
Body Velocity Variance [$\frac{m^2}{s^2}$]	0.1
Yaw Rate Variance [$\frac{rad^2}{s^2}$]	0.05
LiDAR Range Variance [m^2]	0.01
LiDAR Angular Variance [rad^2]	0.005
Horizontal Pos Process Noise [m^2]	0.01
Orientation Process Noise [rad^2]	0.001
Body Velocity Process Noise [$\frac{m^2}{s^2}$]	0.0001
Yaw Rate Process Noise [$\frac{rad^2}{s^2}$]	0.001

6.1.1 S-bend East – No Ambiguities

For the scenario representing no ambiguous geometry, the hash table was generated using a bin size (q_{pose}) of 5 centimeters. In the following, the scenario will be identified as S-bend East, No Ambiguities (SENA). A table listing the total runtime configuration parameters for the data set can be seen in Table 6.4. The map used to seed the hash table generation contained 20 cylindrical features each perturbed by zero-mean Gaussian white noise with a variance of $2.25cm^2$. The perturbation was assumed equal in both the lateral and longitudinal map axes.

Table 6.4: SENA: Parameters

Metric	Value
q_{pose} [m]	0.05
q_{radius} [m]	0.05
b_{limit} [m]	30.00
r_{incl} [m]	60.00
s_{trans}	1.00
Collision Filtering	strict
Descriptors	none

Figures 6.1 and 6.2 show the position and heading of the vehicle first from an overhead view and then as a single dimensional breakdown. From the former, the spread of the landmarks can be seen in relation to the trajectory of the vehicle. In addition, the last subplot of Figure 6.2 shows the speed profile of the vehicle during the maneuver.

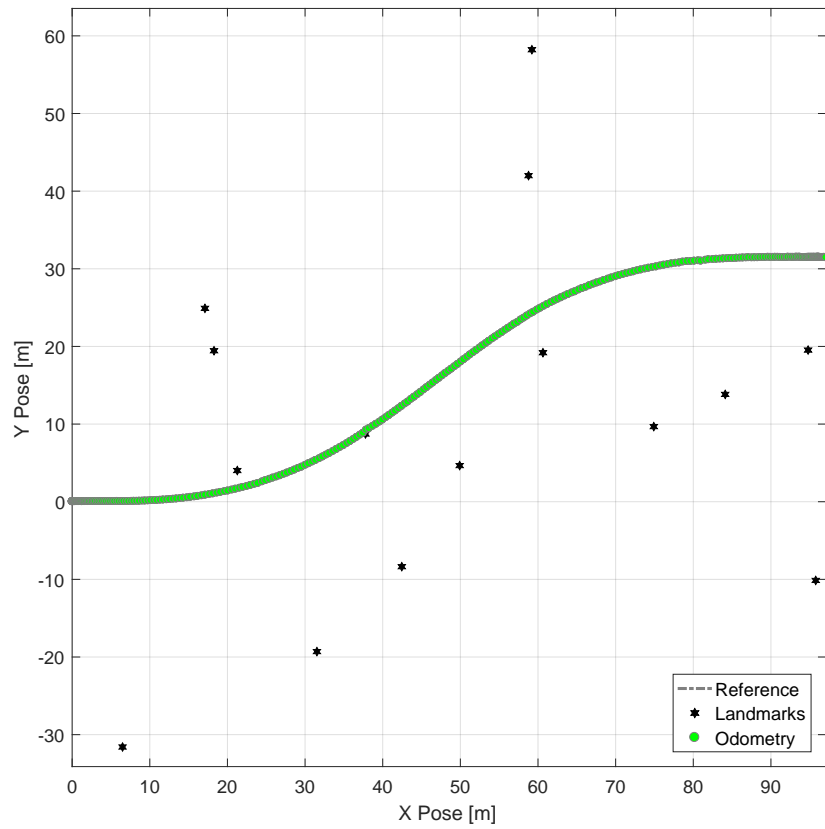


Figure 6.1: SENA: Overlay of odometry with map of landmarks

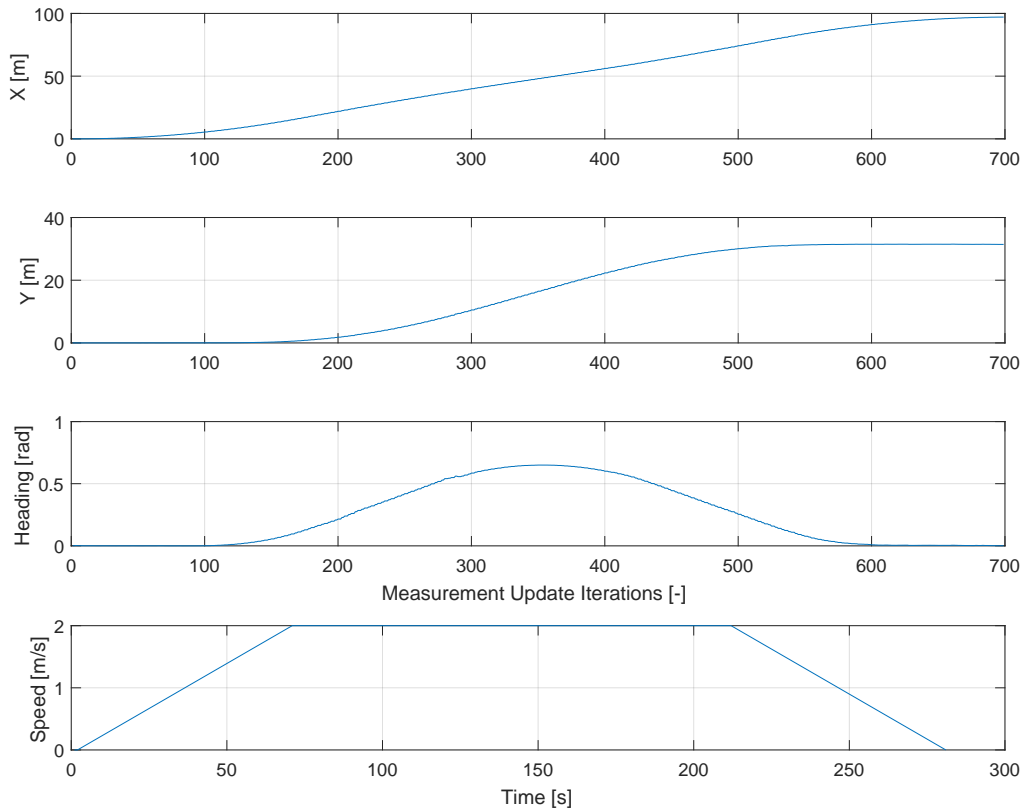


Figure 6.2: SENA: Planar position, heading, and velocity of the vehicle over time

Figure 6.3 shows the error between the actual vehicle path and the localization solution for each dimension. The oscillatory behavior seen in the X and Y dimensions indicates the sampling rate of observations. Figures 6.4-6.5 depict the association results of the data set as a function of each observation sample. Figure 6.4 indicates the status of each recognition step where missed associations are the tally for each observation sample of all detections where no association was found. Incorrect associations represent a tally for each instance where the wrong association was chosen for a detection per observation sample.

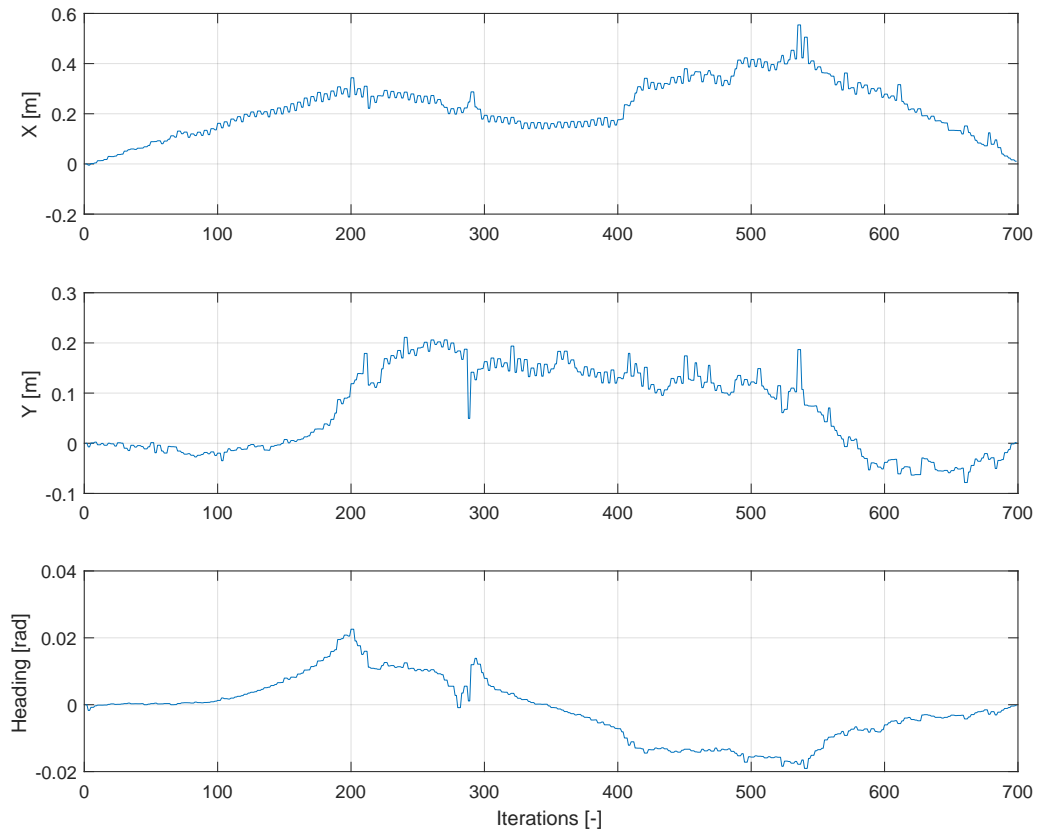


Figure 6.3: SENA: Error between reference solution and odometry

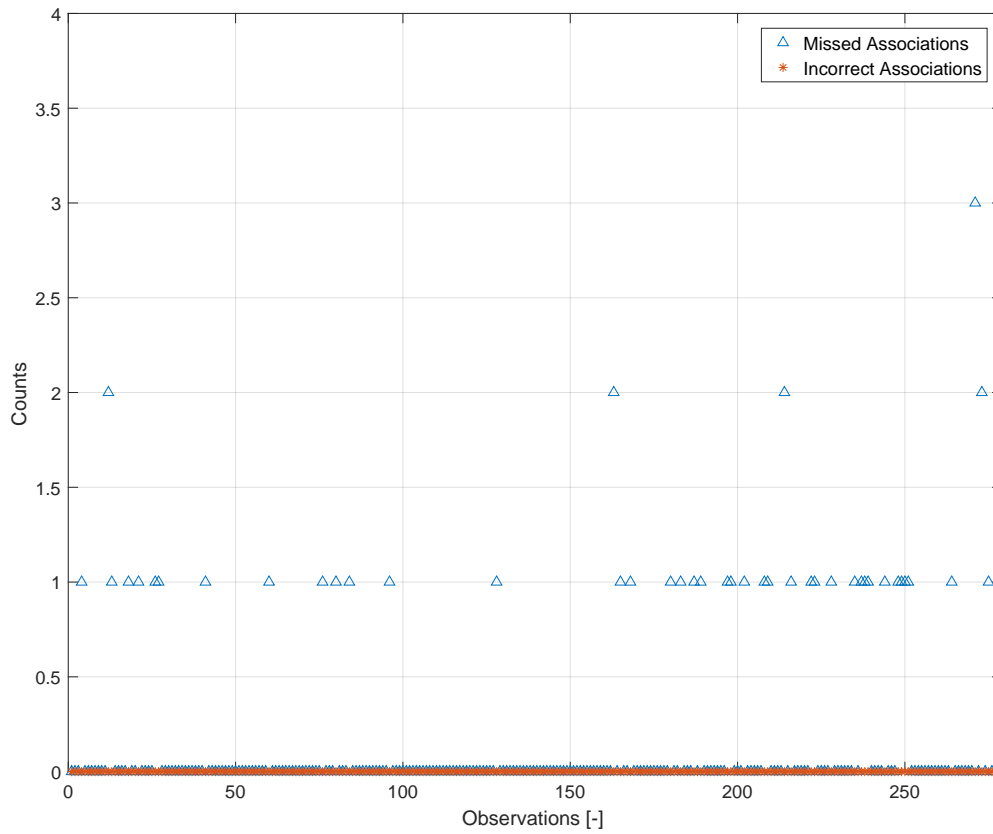


Figure 6.4: SENA: Missed and Incorrect associations at each observation event

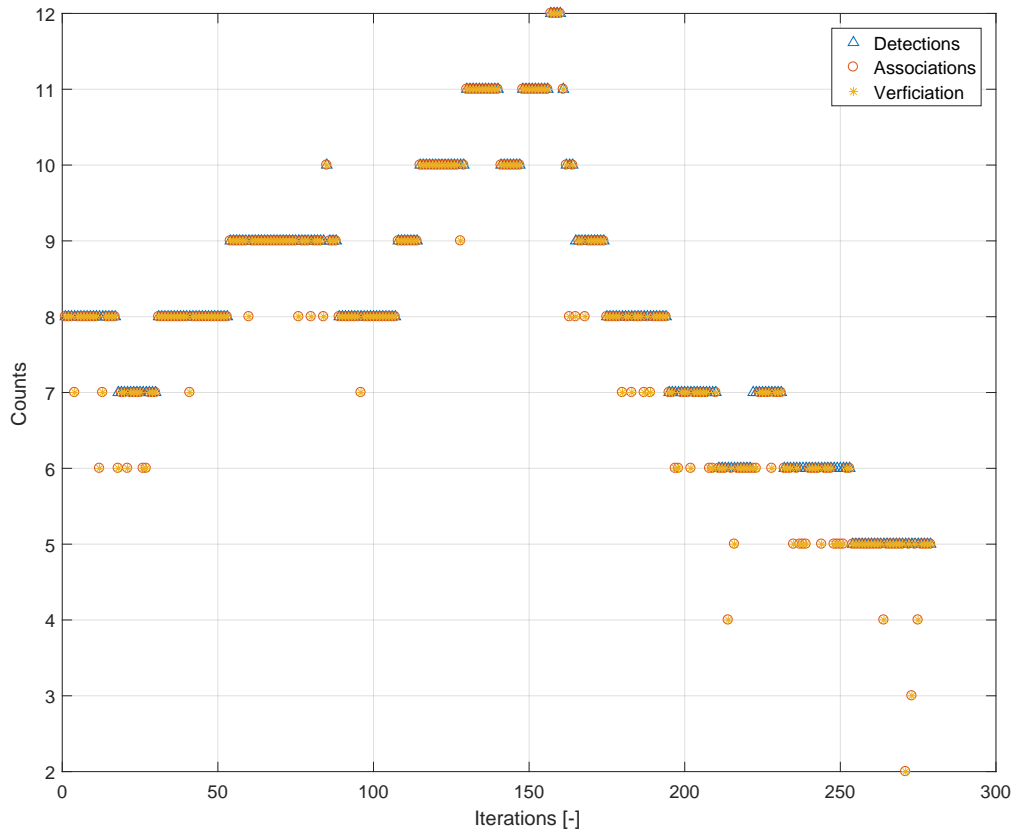


Figure 6.5: SENA: Detections, Associations, and Verifications at each observation event

In Figure 6.5, the blue triangles represent the number of features detected at the corresponding simulation iteration. Likewise, the red circles and yellow asterisks indicate the number of associations and verifications, respectively. In the simulation, verifications are determined from the actual match result that should have taken place. If any feature matches to an incorrect map element, this feature would fail the verification check. When comparing Figure 6.3 and Figure 6.4, it can be seen that missing associations at a given sample does not significantly affect the error. This is only the case because a sufficient amount of correct associations are still achieved for each of these samples as shown in Figure 6.5.

6.1.2 Ambiguity Analysis – Longer S-bend East

In this section, a second simulation scenario is presented to highlight how the system performs under more noise and forced feature symmetry. For the setup, a hash table was generated using a bin size (q_{pose}) of 5 centimeters and the magnitude of variance on the map elements was raised from $2.25cm^2$ to $6.25cm^2$. Additionally, the total amount of map features was raised to 42. A table listing the total runtime configuration parameters for the data set can be seen in Table 6.5. In the following, this scenario will be identified as S-bend East, Longer with Ambiguities (SELA).

Table 6.5: SELA: Parameters

Metric	Value
q_{pose} [m]	0.05
q_{radius} [m]	0.05
b_{limit} [m]	30.00
r_{incl} [m]	60.00
s_{trans}	1.00
Collision Filtering	strict
Descriptors	none

Figures 6.6 and 6.7 again show the position and heading of the vehicle first from an overhead view and then broken down into single dimensional views where the last subplot of Figure 6.7 shows the body speed profile of the vehicle during the maneuver.

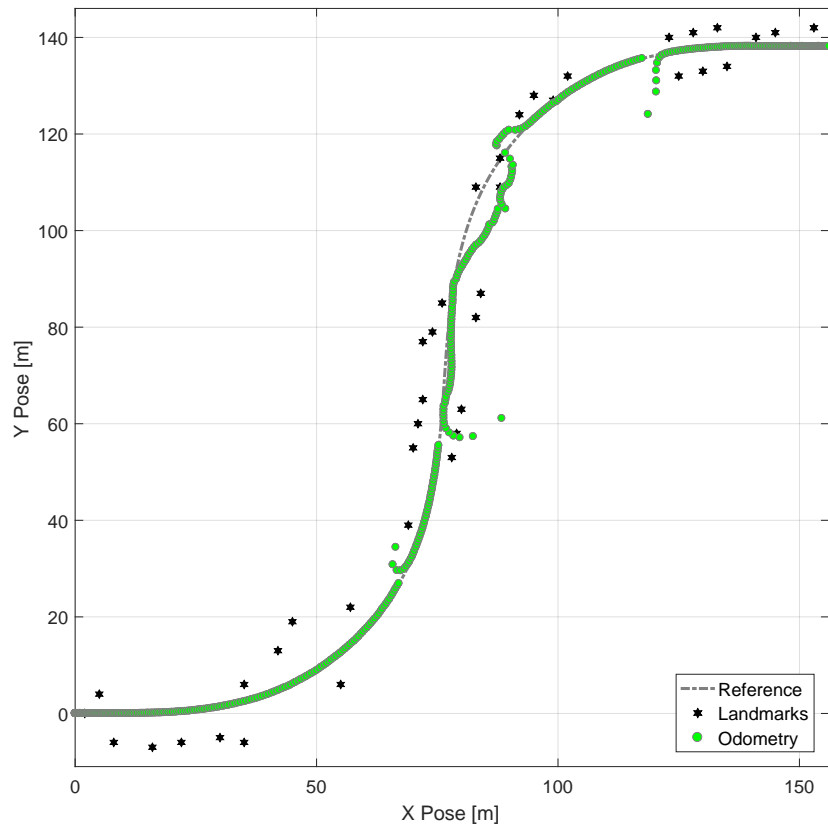


Figure 6.6: SELA: Overlay of odometry with map of landmarks

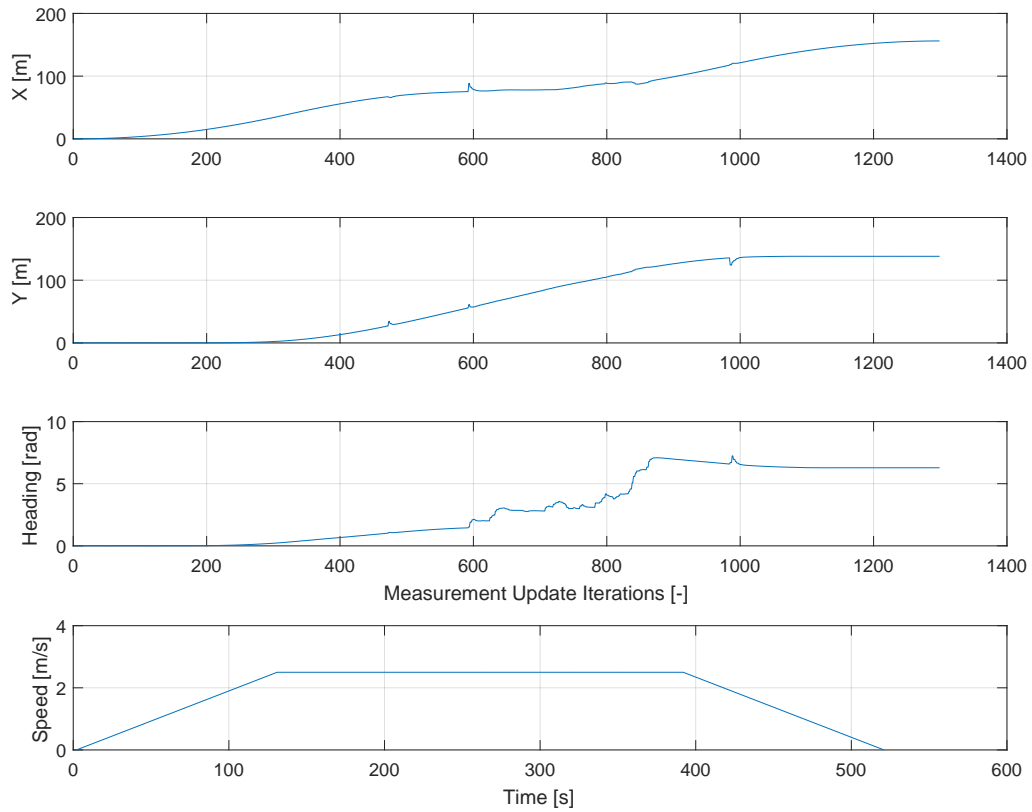


Figure 6.7: SELA: Planar position, heading, and velocity of the vehicle over time

Figure 6.8 shows the error between the actual vehicle path and the localization solution in each dimension. It can be seen that there are severe spikes in error in several places. Figures 6.9-6.10 depict the association results of the data set as a function of each observation sample. A clear relationship can be inferred from each instance of an incorrect association and a spike in the error. Figure 6.9 indicates the status of each recognition step in the same way as shown in the previous data set.

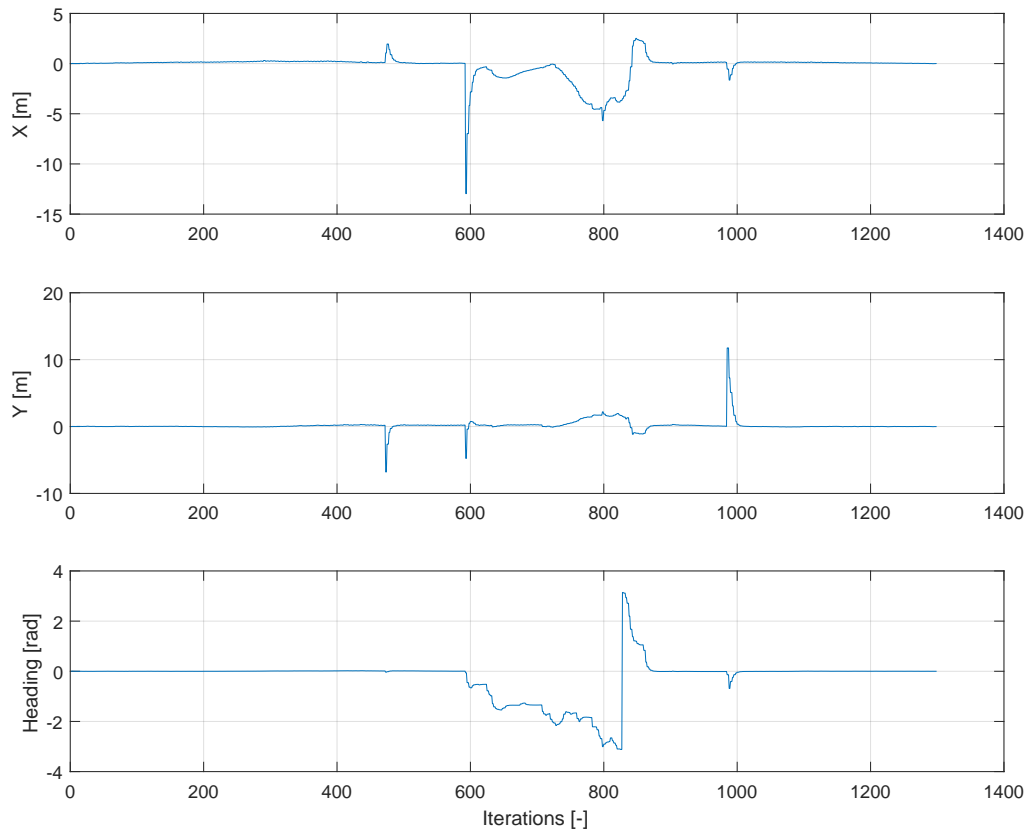


Figure 6.8: SELA: Error between reference solution and odometry

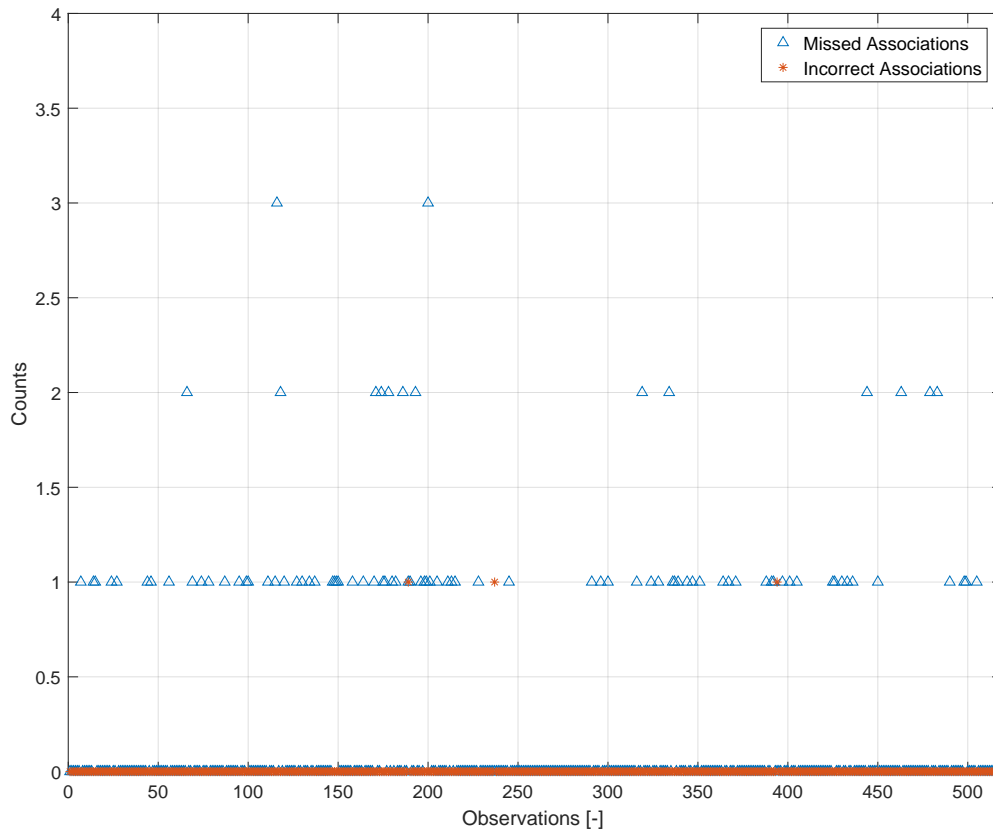


Figure 6.9: SELA: Missed and Incorrect associations at each observation event

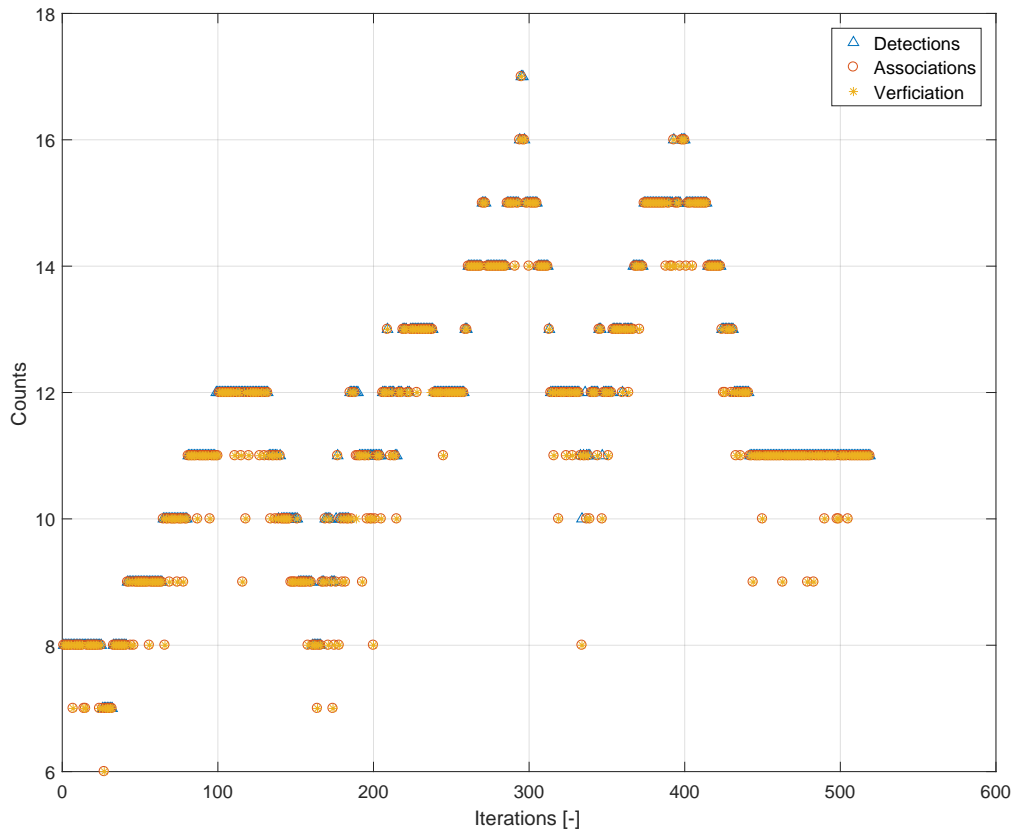


Figure 6.10: SELA: Detections, Associations, and Verifications at each observation event

Figure 6.11 depicts a snapshot of the simulation at observation 189 when an incorrect association occurred. The verified correct associations are circled in blue and the incorrect association is marked with a red cross. The feature that should have been selected in place of the incorrect association is highlighted in an orange box. Lastly, the feature that no association was found for is highlighted by a yellow triangle.

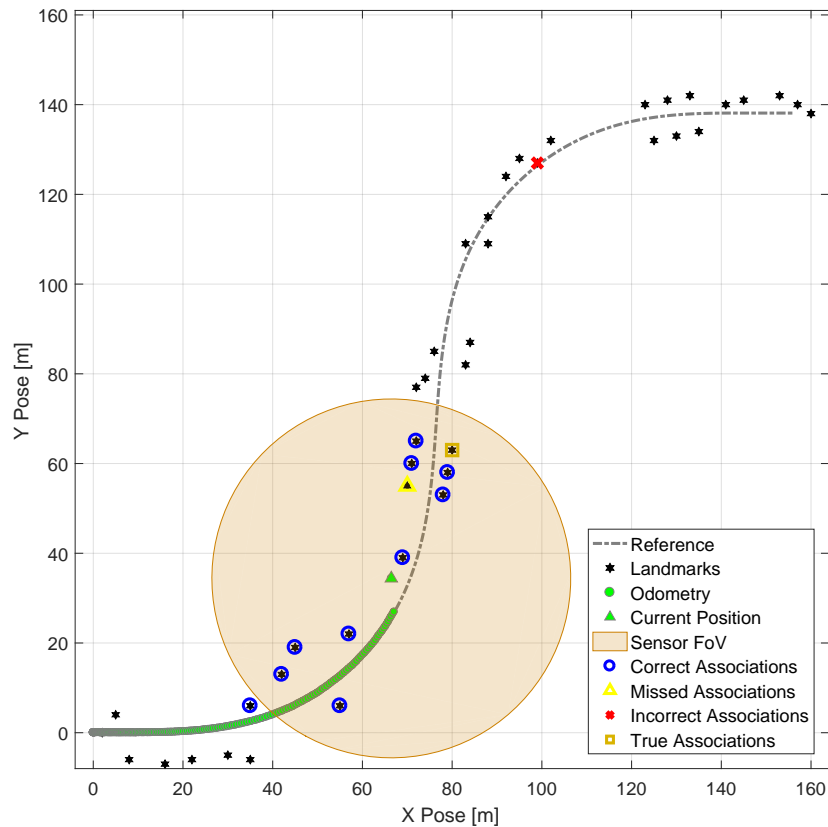


Figure 6.11: SELA: Snapshot of an observation frame where an incorrect association occurred

Since this scenario contains ambiguous feature geometry, some additional information is provided to identify where these ambiguous regions are located as well as how they can affect obtaining the correct association. As mentioned in Chapter 3, ambiguous features are described by a cluster of points from the basis domain that correspond to two or more instances of feature locations in the map domain. Each cluster (or constellation) of points can be roughly positioned by its geometric centroid. In order to understand how many constellations are present along a path, a density heatmap is generated to provide a metric of how ambiguous a region of the map will be during recognition. Figure 6.12 shows the density of ambiguous constellation centroids throughout the entire map.

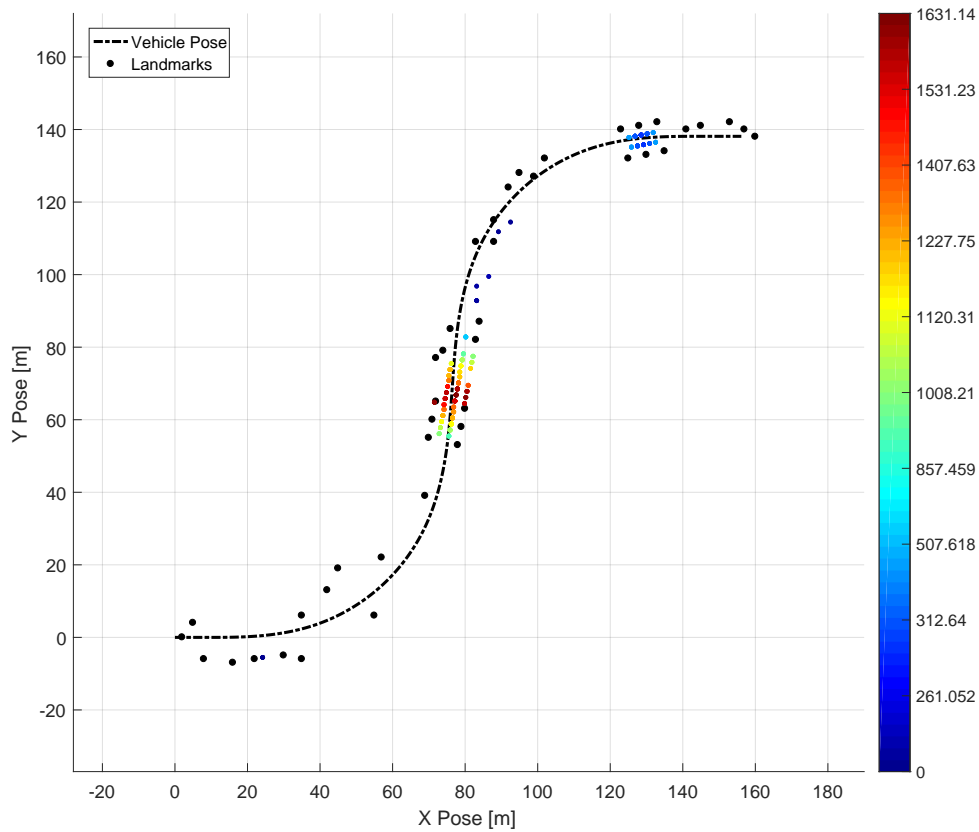


Figure 6.12: SELA: Heatmap representing ambiguous constellation density (by centroid) over mapped area

Figures 6.13-6.14 give an illustration of the ambiguous geometry as it appears in the map sorted by 6 and 3 features, respectively. Each unique color shown in these plots indicate a new ambiguous constellation. Constellations of the same color represent individual occurrences of the ambiguous geometry within the map. In many cases, the number of ambiguous constellations increase as the vertex count decreases. This is largely because the smaller constellations are subsets of the larger ones.

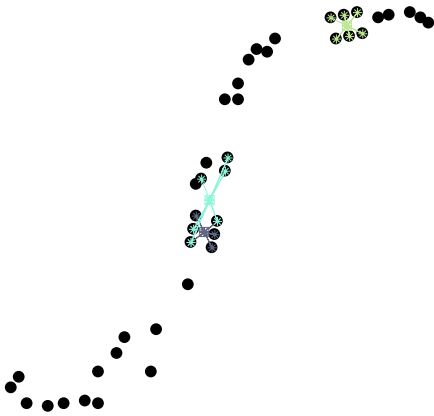


Figure 6.13: SELA: View all ambiguous constellations of 6 vertices

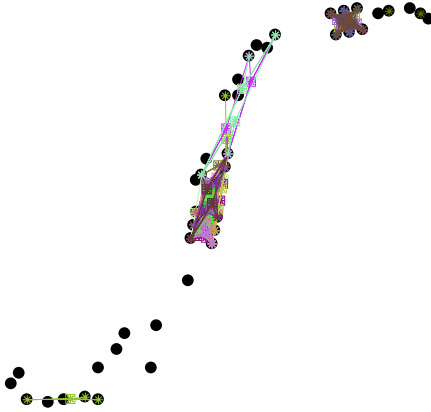


Figure 6.14: SELA: View all ambiguous constellations of 3 vertices

Continuing to build from Figure 6.12, a possible vehicle trajectory can be generated through the mapped terrain to simulate which features may be in view during an online event. With the knowledge of these features along the mock trajectory known, the ambiguities connected to these features are also known thus providing the translational and rotational transformations between each of their occurrences in the map. With all of this information, an analysis of the trajectory is conducted in order to identify regions where choosing the incorrect association due to ambiguities can cause the worst resultant error. Figure 6.15 depicts a heatmap along the trajectory scaled by the minimum and maximum resultant translational offset should an incorrect association occur from an ambiguity. Similarly, Figure 6.16 shows another heatmap scaled by the minimum and maximum rotational offset.

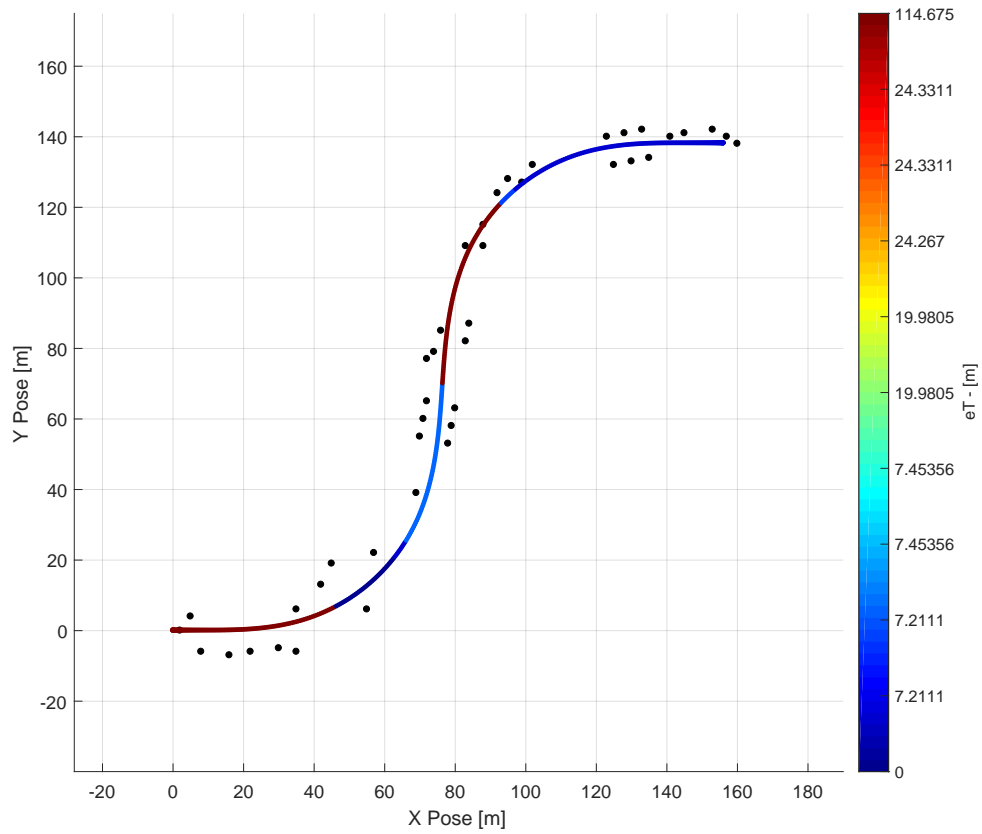


Figure 6.15: SELA: Heatmap representing translational error from ambiguous features over mapped area

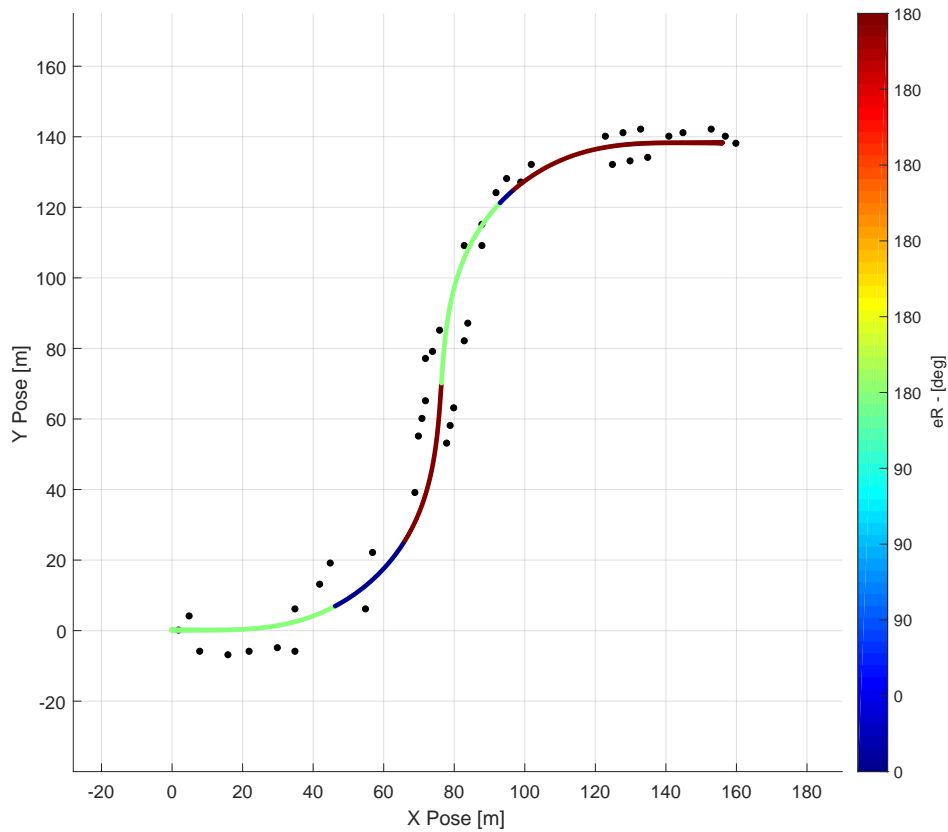


Figure 6.16: SELA: Heatmap representing rotational error from ambiguous features over mapped area

Utilizing the information from the previous set of plots that a high concentration of ambiguous geometry exists in the middle of the map, an expectation is set that the recognition step may struggle in this region. A correlation to this can be seen in Figure 6.17 which depicts a measure of probability for the correct association given the presence of ambiguous features. This probability is shown to be 1 in regions where no features that belong to ambiguous constellations are observed and shown to approach 0 as the vehicle enters an ambiguous area. The red triangles in Figure 6.17 indicate the exact sample where an incorrect association was chosen aligning with Figure 6.9 as shown previously.

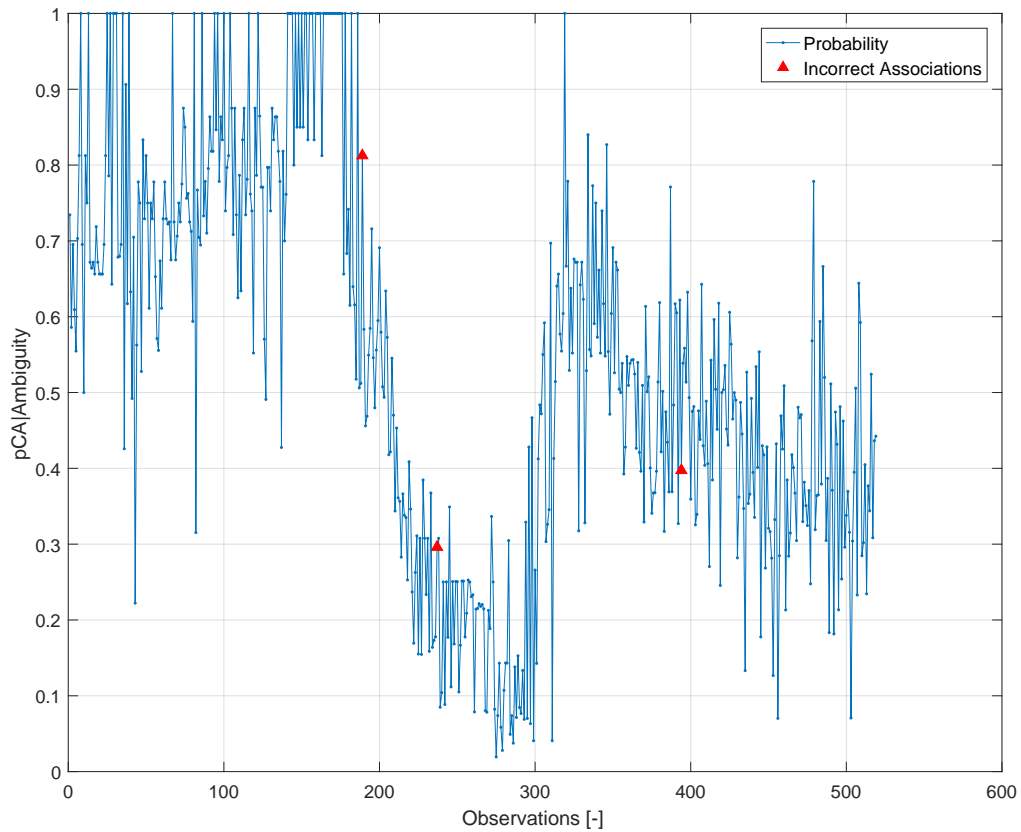


Figure 6.17: SELA: Probability of correct association given ambiguous features

Along a similar idea, Figure 6.18 presents a metric of probability for the correct association given all the candidate layers for the observation. In other words, this metric shows how many layers specifically provided evidence for the chosen association. Again, the red triangles indicate the sample where an incorrect association was chosen for this scenario.

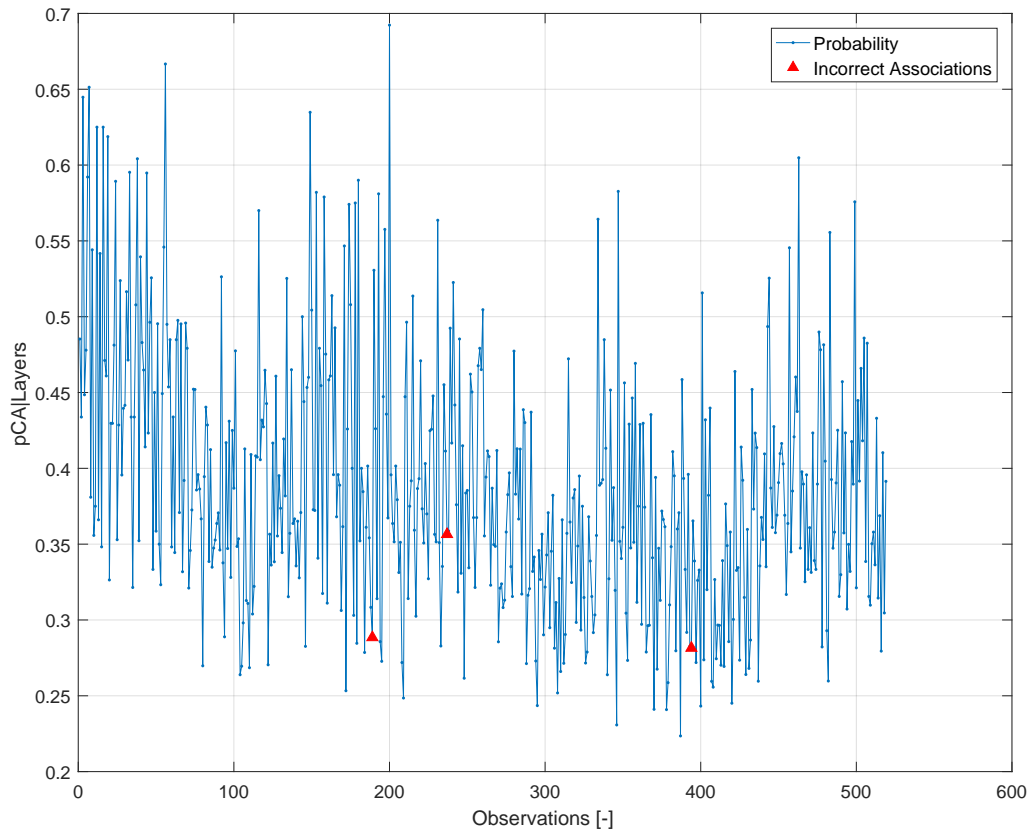


Figure 6.18: SELA: Probability of correct association given layer candidates

6.1.3 Discussion of Results for S-bend East

The numerical results of the simulation runs can be seen in Tables 6.6-6.7 and Tables 6.8-6.9 where the former indicates the total RMS error and the latter shows statistics about the online feature extraction and matching for the duration of the run. Odometry, in the case of these results, represents the positioning solution at each sample. Since there are some prediction updates where no measurements are available, the solution during these samples is purely the integrated position of the vehicle based on input. When measurements become available, the prediction is corrected with the measurements for a closed-loop update.

Table 6.6: SENA: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Odometry	0.2498	0.1059	0.0094

Table 6.7: SELA: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Odometry	1.2543	0.8893	0.8249

Table 6.8: SENA: Result statistics

Metric	Value
Percent Associations	97.7589
Percent VerifiedAssociations	100.0000
Average Detections	7.9964
Average Associations	7.8172
Average VerifiedAssociations	7.8172

Before starting the results discussion, it is important to note a few additional caveats in the simulation that differ greatly from what the system would normally experience in a real world scenario. First, the mapped feature elements are generated pseudo-randomly which severely decreases the likelihood of encountering ambiguous geometries. It is possible to have elements

Table 6.9: SELA: Result statistics

Metric	Value
Percent Associations	98.2146
Percent VerifiedAssociations	99.9495
Average Detections	11.6551
Average Associations	11.4470
Average VerifiedAssociations	11.4412

of randomness in real-world maps, but typically urban environments have a great deal of symmetry which hinders the matching procedure. Next, all features in the simulation that can be seen within the simulated LiDAR FoV, even if occluded, will still produce a range-bearing measurement. This means that the maximum amount of possible information will always be provided to the system each measurement sample. Lastly, there are no simulated false positive or false negative detections that can stress the matching algorithm and all feature measurements arrive in a batch rather than asynchronously. Batching measurements makes applying the correction back to the states much easier since all the individual detections have the same timestamp.

On a first glance, the results presented in Table 6.8 from this simulation run are very good. The total RMS error turned out well beneath the meter-level accuracy goal. Moreover, tracing the error over the duration of the run indicated that the solution stayed within 0.5 meters, thus indicating good precision. It is interesting to note that lateral error remained low during times of acceleration and performed worse in times of constant velocity. Longitudinal error was subject to the opposite affect, while heading performed worst during the apex of curvature along the path.

As expected, the second scenario that included ambiguities performed worse. Over the duration of the maneuver, the solution selected an incorrect association 3 times which causes a spike in error and took time to return back to steady state. Rationally, achieving the poor resultant RMS error while maintaining a nearly perfect run of associations demonstrates that the positioning solution is extremely sensitive to disturbances. With that aside, it is also important

to note that during each of the instances where an incorrect association occurred, it was not a result of an ambiguity. The cause of each incorrect association is a result of a combination of measurement noise and hash space quantization boundaries. This causes a feature to fall into a bin that matches a different layer in the hash table than it was hashed at. This is a phenomenon that is not captured in any of the ambiguity integrity risk plots shown in Section 6.1.2. However, even though the incorrect association was not mangled by an ambiguous constellation, it does not indicate that ambiguities don't play a role. There are many instances as shown in Figure 6.9 where associations could not be determined for a given detection. Put simply, this could be a result where insufficient evidence was provided to suggest a clear favorite candidate during matching. Despite this, the probability of a correct association given the presence of ambiguities shown in Figure 6.17 is reasonably low in areas indicating that the recognition procedure is fairly robust to the influence of ambiguities.

6.2 Integrated Localization Solution

The following sections in this chapter will cover the results collected from integrating Geometric Hashing Localization into the pipeline from *Kümmerle et al*[23]. For each circuit introduced in Chapter 5, both a coarse and fine resolution procedure was recorded according to different values for the quantization parameter. For all data sets, it was found that Algorithm 4.7 yielded the most consistent results for analysis.

Since incremental state updates to Geometric Hashing Localization do not depend on prior system states, there is very little inertia preventing the solution from bouncing around. For this reason, quantitative results for the localization solution are presented using a convergence rule. The solution is considered to be converged at any sample where the positional error from the reference solution is sufficiently small and if the solution is reasonably smooth within a given window. To obtain a metric of smoothness, the root-mean-squared deviation (RMSD) algorithm was used in a sliding window. The window size for each data set was selected as a function of the total upsampled length of the data. In all cases presented here, this function was 0.5% of the data length. For the convergence tolerance parameters, the system was considered to be converged in the horizontal position if the error was less than 10 meters and the RMSD is less

than 1 meter. Similarly, for system heading, convergence was attained if the error was less than $\frac{\pi}{8}$ radians and the RMSD is less than $\frac{\pi}{12}$ radians. Total system convergence is reported if and only if the system is converged in both position and orientation. It is also important to note, that convergence is reported on a per sample basis. In other words, if the system is converged and a new sample arrives that violates the convergence rule, the new sample is considered diverged.

For each data set, two tables will be presented: one reporting the total RMS error for various signals, and the other highlighting several statistical metrics useful for comparing data sets against one another. The odometry reported in the remaining tables and figures is slightly different from Section 6.1.3. The difference is due to the concept that measurements are not fused via a Kalman filter approach. Instead, as mentioned in Section 5.5, all information at each sample (with or without associated feature identifiers) is optimized over for the best possible positioning solution. Next, the vehicle positions (X, Y, and Heading) are provided by the reference solution, the GNSS-only solution, and the Geometric Hashing Localization solution as well as the error between the reference and Geometric Hashing Localization solution. The plots showing system error (Figures 6.24, 6.32, 6.45, and 6.53) indicate regions where the solution is considered to be converged or not. Following these, are two plots pertaining to the quality of feature extraction and data association steps. The first plot, named system availability, provides an indication of how many features were detected at each update sample. For each of those updates, the number of associations is also reported. Each association is considered to be valid if the conditions presented in Section 4.2.3 hold true (*i.e.* features are in close proximity and within a reasonable range of the prior position). The second plot in this series depicts the correlation of system processing time with the number of detected features. Finally, the remaining four plots in each data set breakdown each update sample relating position to the results of each feature matching attempt. There are four possible conditions for each update sample:

- No Detections - No features could be extracted at this position, only odometry used,
- No Associations - Features are present, but no matches could be found, only odometry used,
- No Verification - Matches found, but verification failed, only odometry used,

- Verification Success - Matches found and verified, odometry and matched features used

6.2.1 Sindelfingen

For each data set collected on the Sindelfingen circuit, the hash tables were generated using a map of 738 cylindrical features assumed to be perturbed by Gaussian White Noise of a variance of 25cm^2 . The perturbation was assumed equal in both the lateral and longitudinal map axes. In addition to the hash values representing 2D positions, the recognition step utilized the radius of the cylinders as an additional matching descriptor.

6.2.1.1 Fine Resolution – Sifi5

For the course resolution data set on the Sindelfingen circuit, the hash table was discretized using a quantization parameter q_{pose} of 5 centimeters and collisions were also dealt with using the strict filter. Additionally, no bases were created from features separated further than 60 meters (b_{limit}) and all features outside a 100 meter radius from each basis origin (r_{incl}) were excluded from the layer data. Additional runtime parameters can be seen in Table 6.10. Features were matched using Algorithm 4.7 as presented in Chapter 4. The hash table produced from these parameters contained 2,571,008 hash entries and comprised of 37,236 layers resulting in a file size of 420 MB. In the following, this scenario will be identified as SIFI5.

Table 6.10: SIFI5: Parameters

Metric	Value
q_{pose} [m]	0.05
q_{radius} [m]	0.05
b_{limit} [m]	60.00
r_{incl} [m]	100.00
s_{trans}	1.00
Collision Filtering	strict
Descriptors	none

Figure 6.19 shows the path of the vehicle along with all the landmarks extracted from the circuit presented previously in Figure 5.3. A breakdown of the solution state at each update is also depicted to indicate areas where the solution was unable to perform as expected. As the legend explains, pink "X" symbols indicate sample updates where no landmark detections were found. Yellow triangles show updates where landmarks were detected, but insufficient associations were found to utilize the data. Blue squares indicate areas where matches were found, but did not pass the verification procedure whereas green circles met all necessary criteria to report a positioning correction to the solution. Figures 6.20, 6.21, and 6.22 all utilize the same legend, but show the positioning solution as an individual dimension over time.

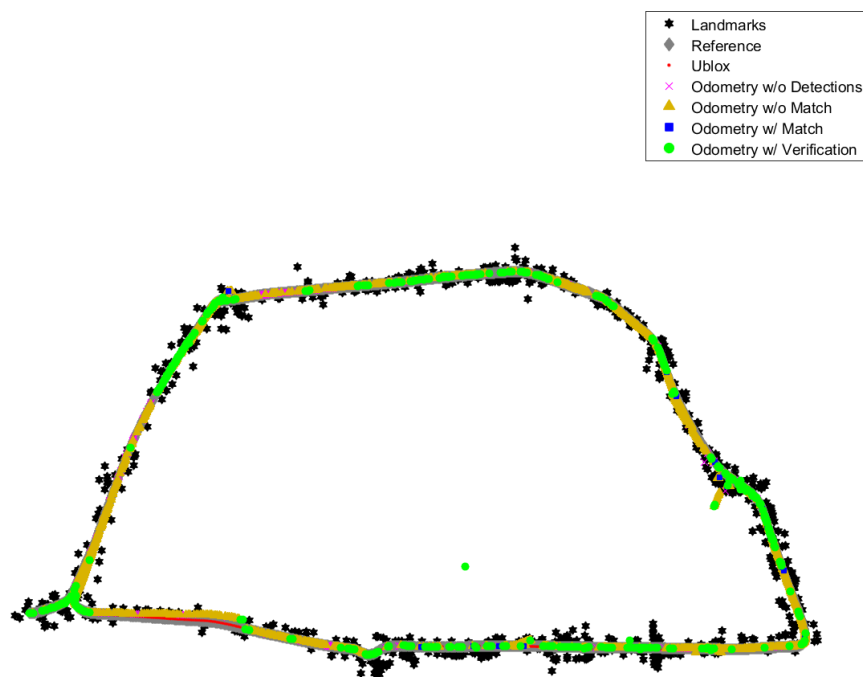


Figure 6.19: SIFI5: Overlay of odometry with map of landmarks

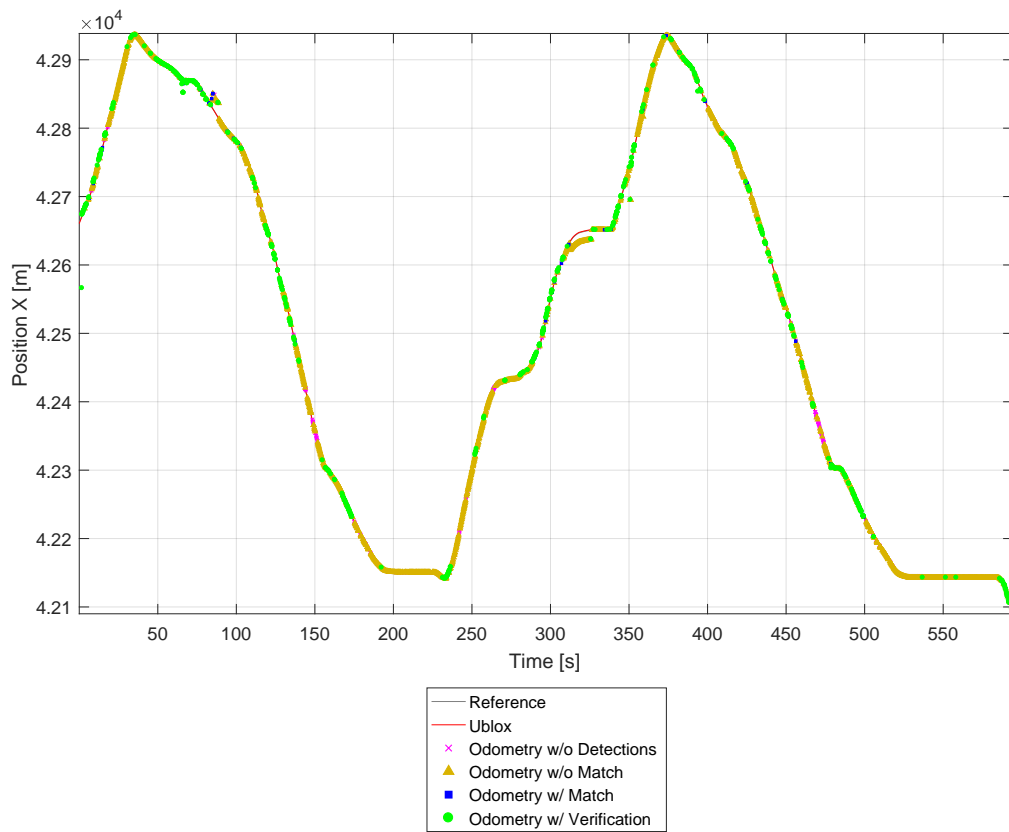


Figure 6.20: SIF15: Breakdown of X position with update availability

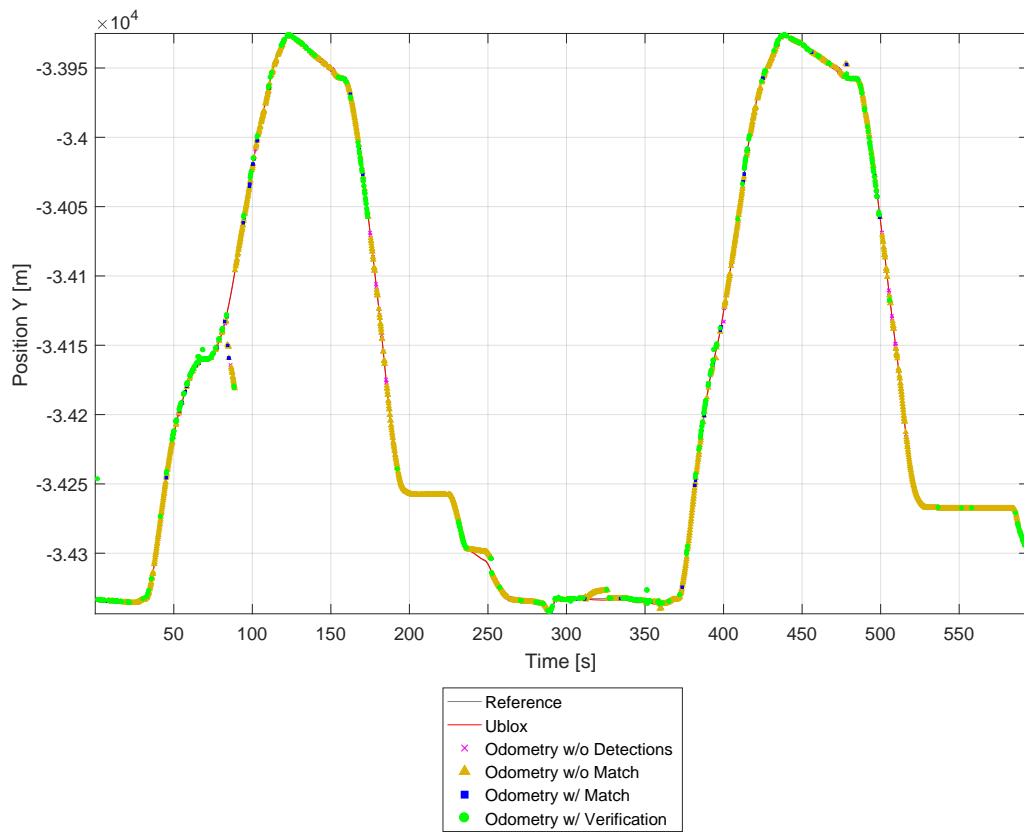


Figure 6.21: SIFI5: Breakdown of Y position with update availability

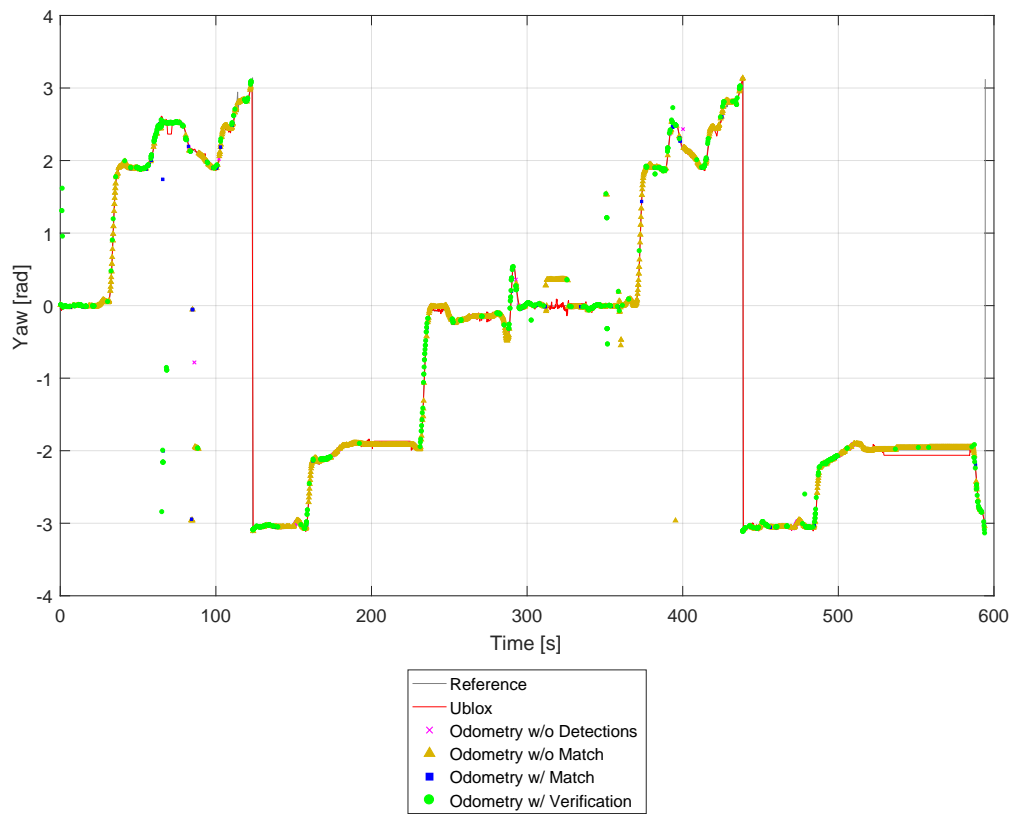


Figure 6.22: SIFI5: Breakdown of heading with update availability

Figure 6.23 shows the horizontal positioning and heading solutions from the data set. As mentioned above, the reference solution is the result of the algorithm presented in Section 5.2. The data marked with "Ublox" is a Global Navigation Satellite Systems (GNSS) solution fused with Inertial Navigation Systems (INS) solution only. Lastly, the data marked "Odometry" is the Geometric Hashing Localization solution.

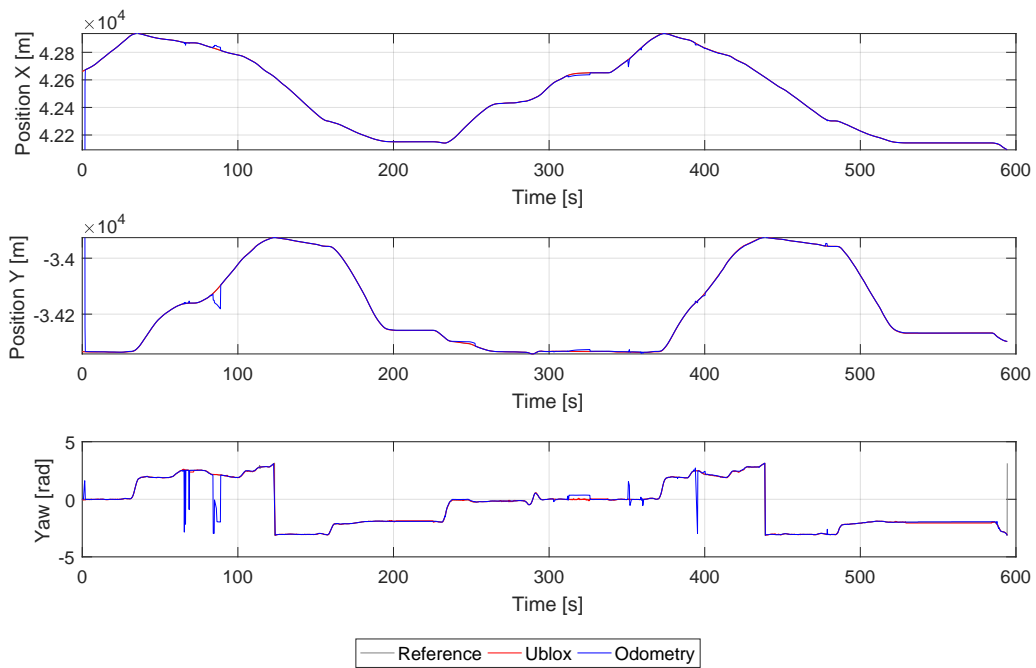


Figure 6.23: SIFI5: Planar position and heading of the vehicle over time

Figure 6.24 depicts the error between the Geometric Hashing Localization solution and the reference solution and is segregated according to the convergence rules presented in Section 6.2 where red indicates the raw error and blue only represents a converged result.

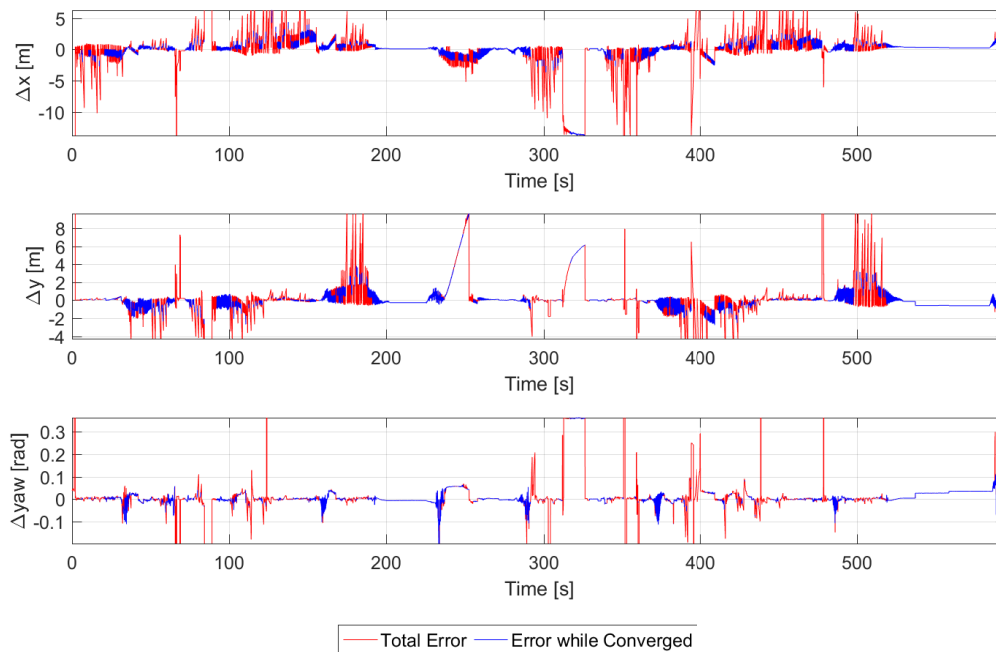


Figure 6.24: SIFI5: Error between reference solution and odometry

Figure 6.25 shows association results of the solution. Black lines indicate the total number of detected features for the given sample. Red and green lines each indicate the number of associations for the sample. In areas where black is clearly visible, there exist no associations. Green represents areas where associations were considered valid whereas red lines failed the matching process at some stage. Lastly, a relationship between the number of extracted features and the total processing time of the update is depicted in Figure 6.26.

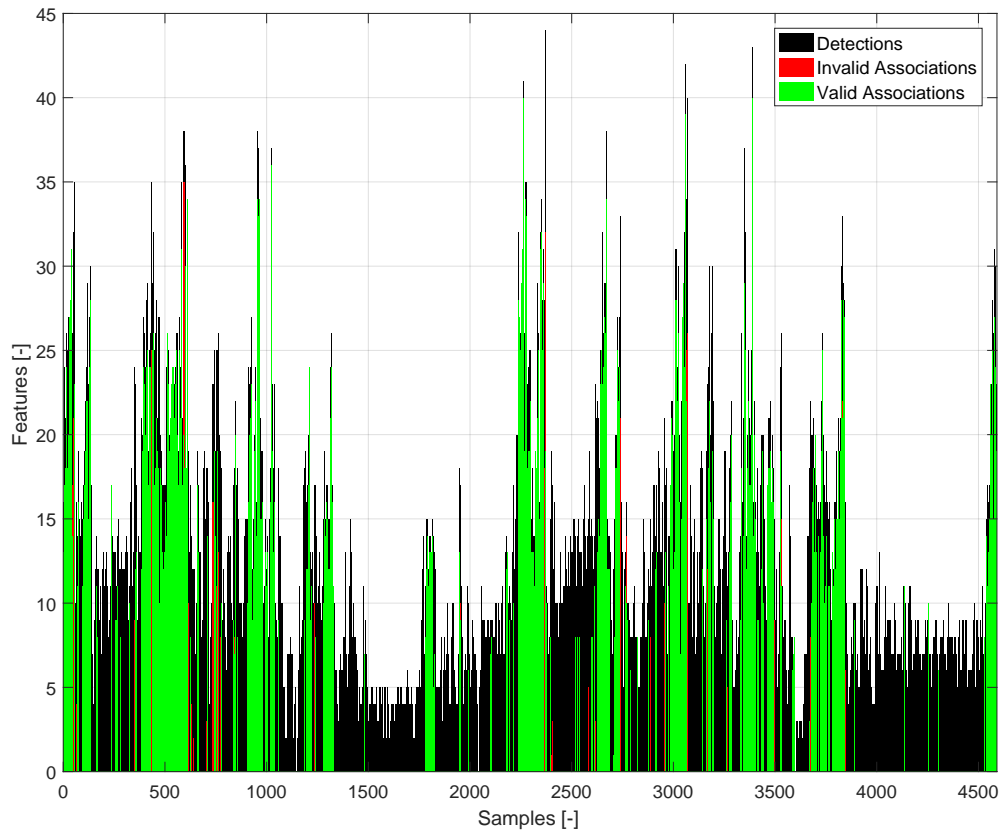


Figure 6.25: SIFI5: Measurement correction availability during the drive

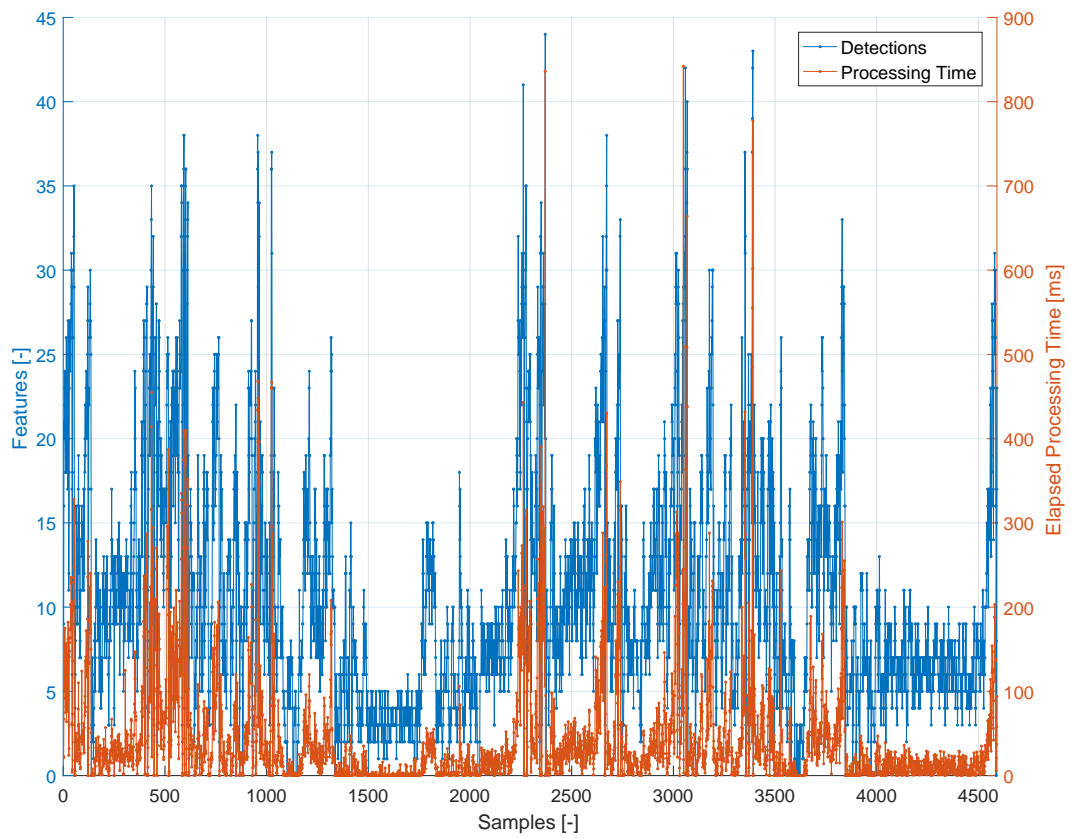


Figure 6.26: SIFI5: Processing time for each detection during the drive

6.2.1.2 Coarse Resolution – Sifi20P

For the coarse resolution data set on the Sindelfingen circuit, the hash table was discretized using a quantization parameter q_{pose} of 20 centimeters and collisions were also dealt with using the strict filter. Additionally, no bases were created from features separated further than 60 meters (b_{limit}) and all features outside a 100 meter radius from each basis origin (r_{incl}) were excluded from the layer data. Additional runtime parameters can be seen in Table 6.11. Features were again matched using Algorithm 4.7 as presented in Chapter 4. The hash table produced from these parameters contained 666,532 hash entries and comprised of 37,236 layers resulting in a file size of 254 MB. In the following, this scenario will be identified as SIFI20.

Table 6.11: SIFI20: Parameters

Metric	Value
q_{pose} [m]	0.20
q_{radius} [m]	0.05
b_{limit} [m]	60.00
r_{incl} [m]	100.00
S_{trans}	1.00
Collision Filtering	strict
Descriptors	none

Figure 6.27 again shows the path of the vehicle along with all the landmarks extracted from the circuit presented in Figure 5.3. Figures 6.28, 6.29, and 6.30 all utilize the same legend as was previously stated.

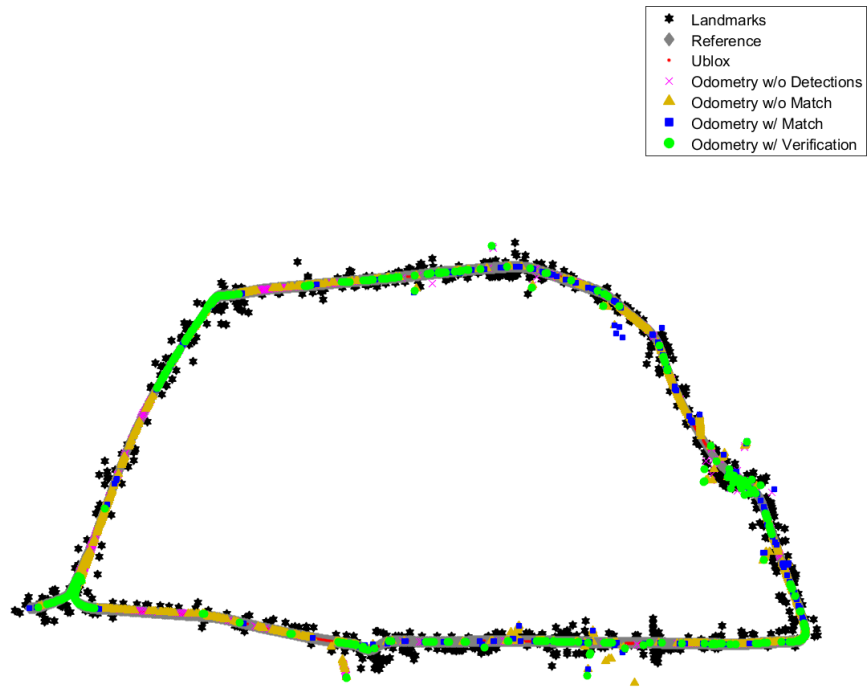


Figure 6.27: SIFI20: Overlay of odometry with map of landmarks

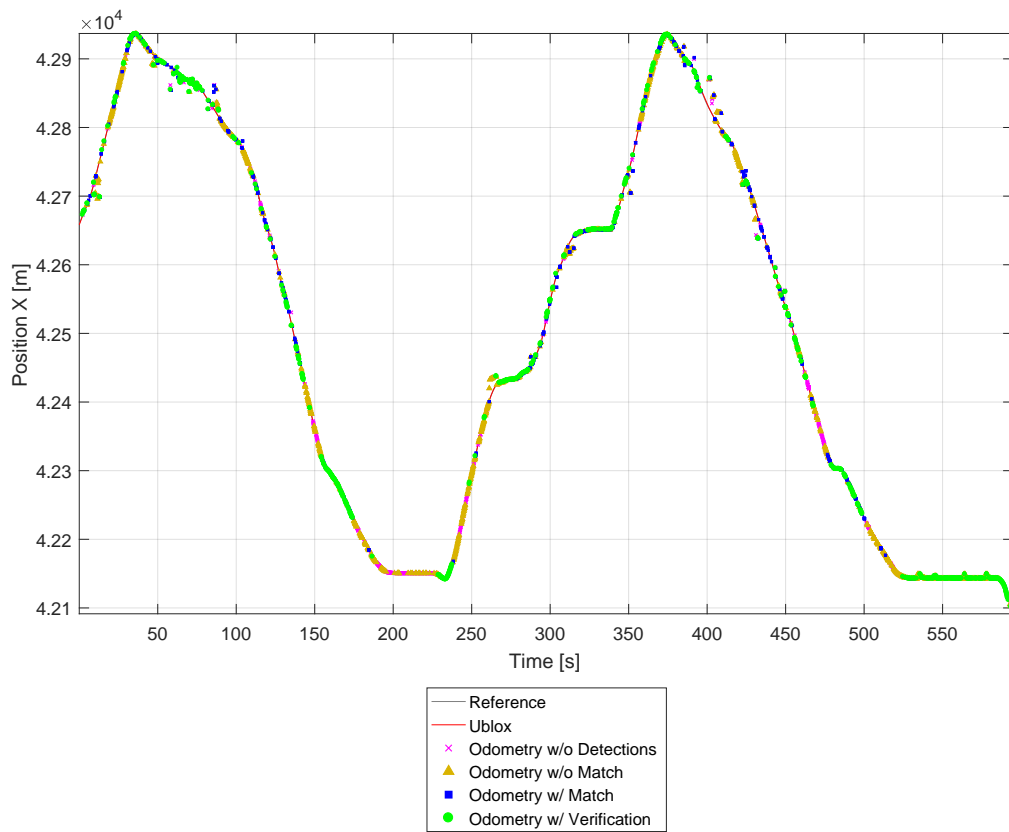


Figure 6.28: SIFI20: Breakdown of X position with update availability

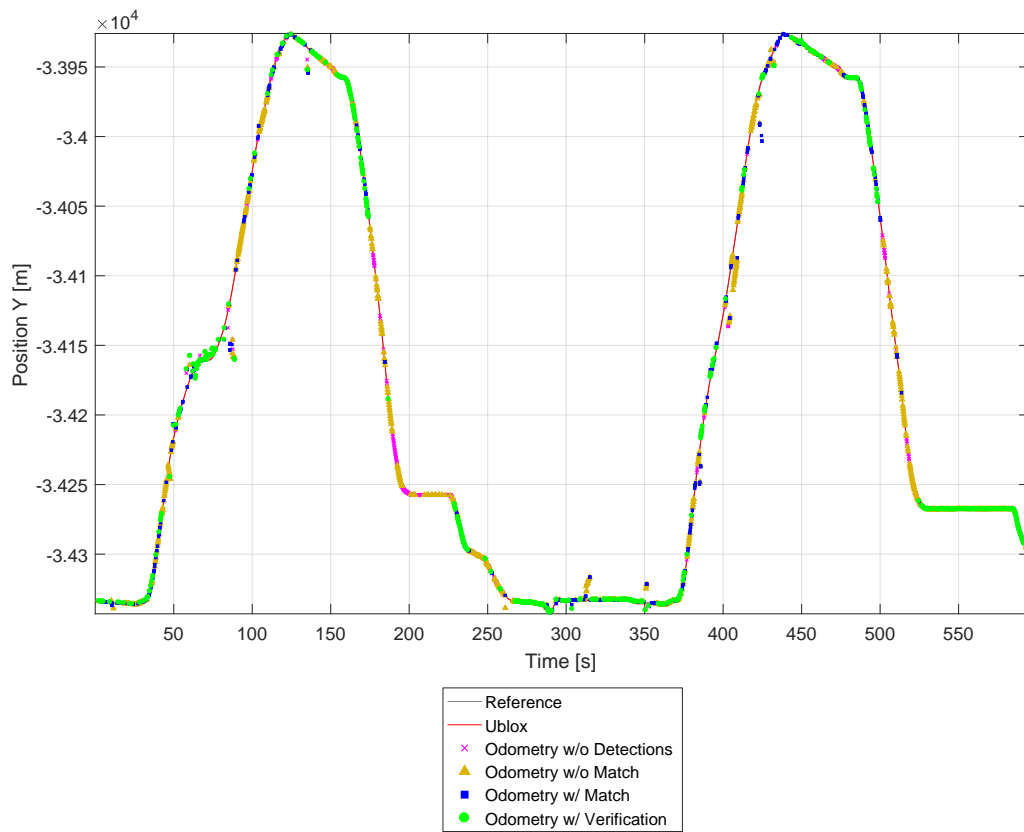


Figure 6.29: SIFI20: Breakdown of Y position with update availability

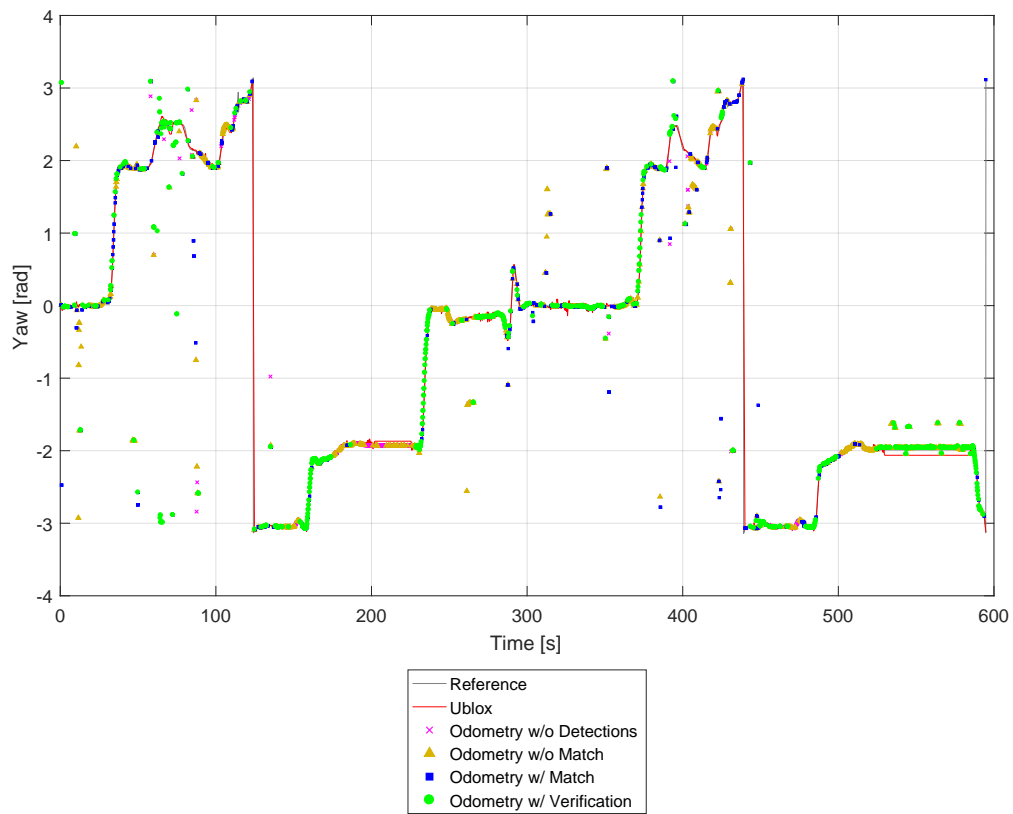


Figure 6.30: SIFI20: Breakdown of heading with update availability

Figure 6.31 again shows the horizontal positioning and heading solutions from the data set similarly to 6.23. Figure 6.32 again depicts the error between the Geometric Hashing Localization solution and the reference solution and is segregated according to the convergence rules presented in Section 6.2.

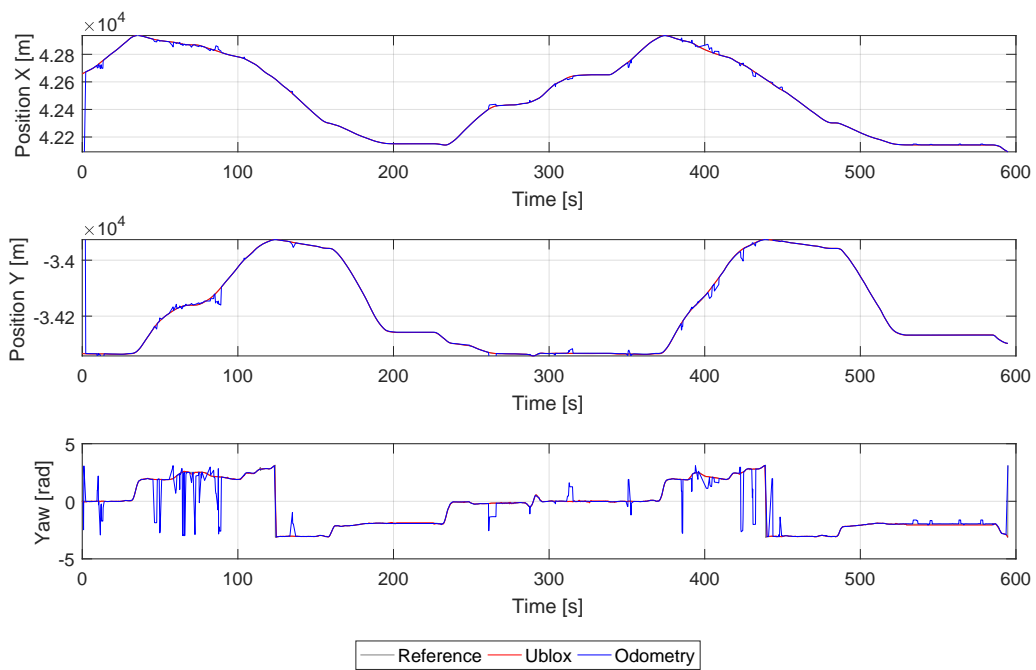


Figure 6.31: SIFI20: Planar position and heading of the vehicle over time

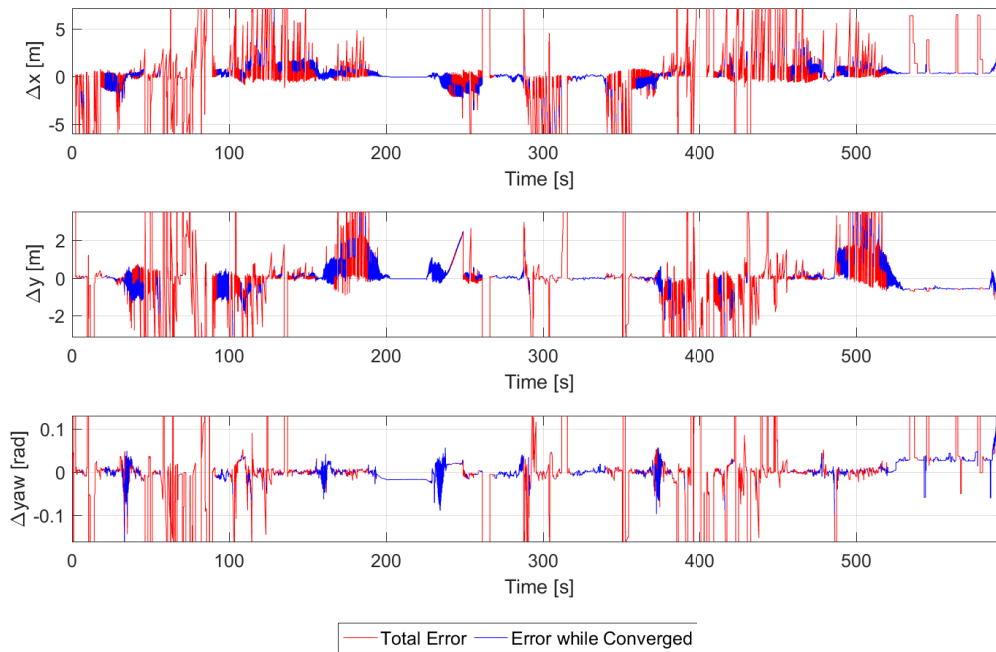


Figure 6.32: SIFI20: Error between reference solution and odometry

The association results of the solution using the same format as Figure 6.25 are shown in Figure 6.33 for the coarse run. Lastly, figure 6.34 depicts a relationship between the number of extracted features and the total processing time of the update.

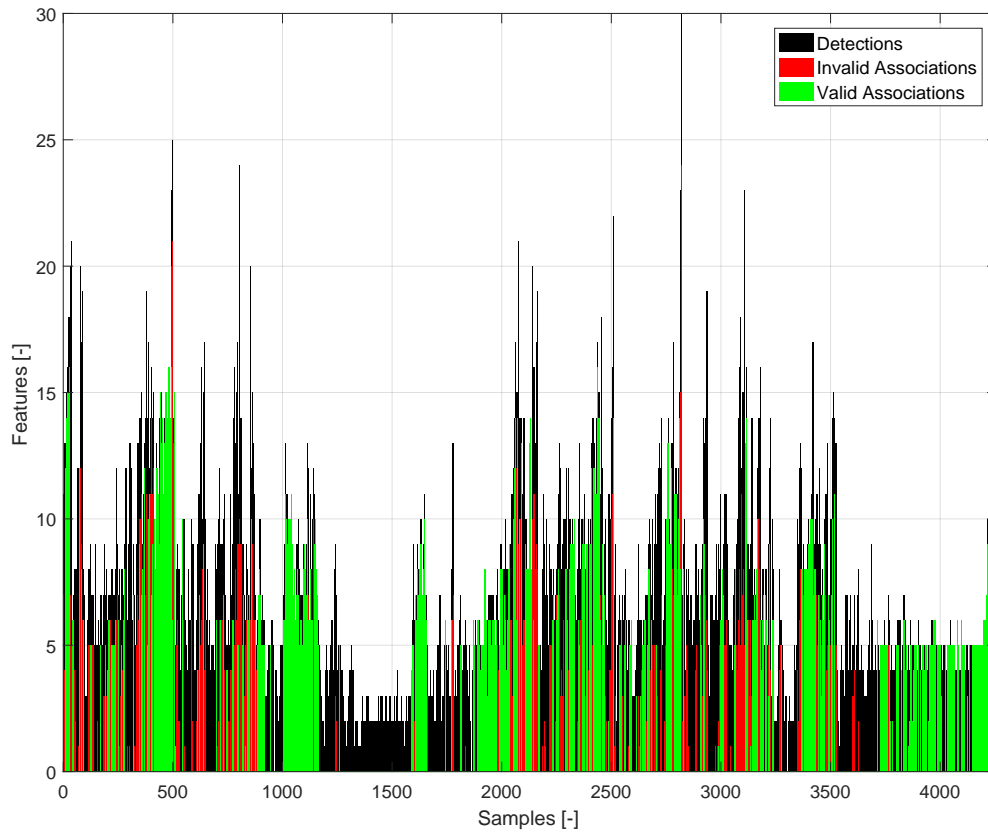


Figure 6.33: SIFI20: Measurement correction availability during the drive

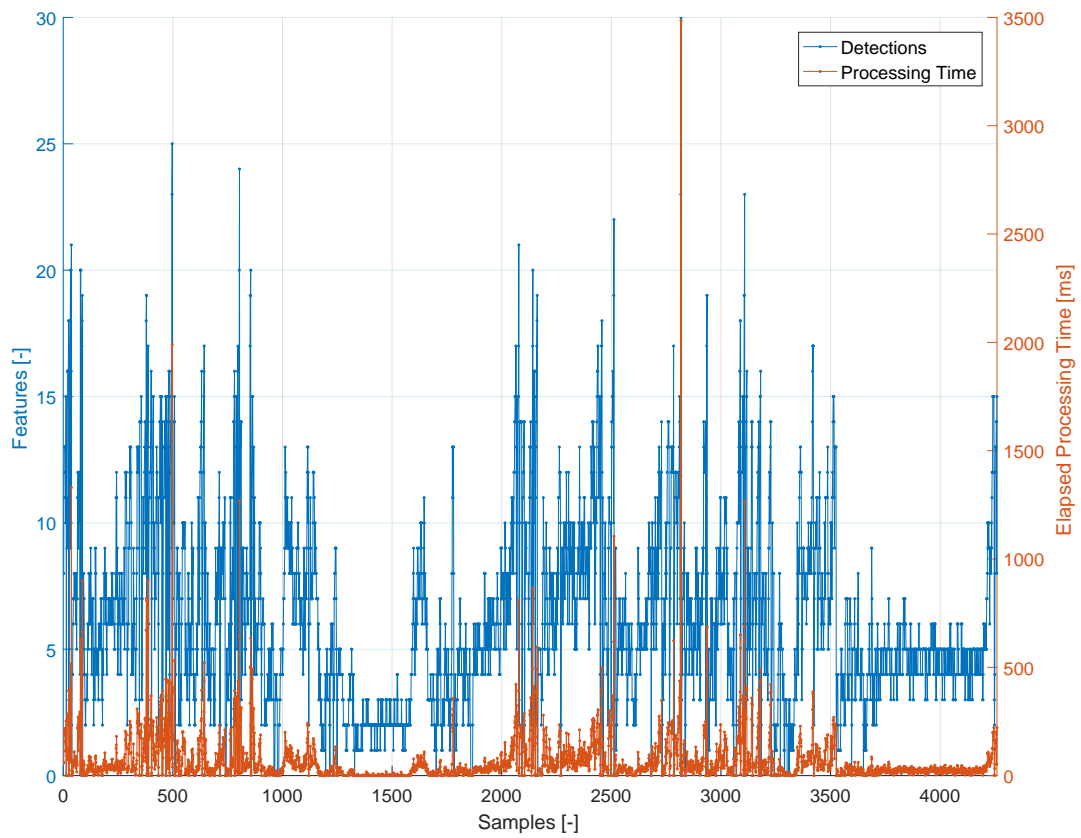


Figure 6.34: SIFI20: Processing time for each detection during the drive

6.2.1.3 Discussion of Sindelfingen Results

As expected after analyzing the results of the simulated experiments shown in Tables 6.12-6.13, the coarse data set provided results with lower RMS error values when compared to the fine data set when the system was converged, but had a lower percentage of samples within the convergence envelope. The time spent converged introduces a new variable that could not be studied with the simulated data and inherently correlates heavily with the frequency of correct associations. Unlike the simulated scenario, the Sindelfingen data set is rich with ambiguous cylindrical feature geometries as can be seen in Figures 6.35-6.36. The former plot shows the density of ambiguous constellation centroids similar to Figure 6.12 while the latter shows the locations and occurrences of ambiguous constellations of 3 vertices in the map. Just like Figure 6.14, each color represents a unique ambiguous constellation and multiple instances of the same color depict the locations of those constellations within the map. In addition, there are also several pockets of very sparse, or even the complete absence of, features within the FoV of the vehicle's sensors. This produces additional challenges to the recognition and validation step.

Table 6.12: SIFI5: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Ublox	1.8246	1.3533	0.1901
Odometry	2048.3984	1649.0362	0.4535
Odom Converged	2.3034	1.4338	0.0616

Table 6.13: SIFI20: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Ublox	3.0019	2.2625	0.3182
Odometry	1606.0553	1296.1178	0.8076
Odom Converged	0.8347	0.6000	0.0203

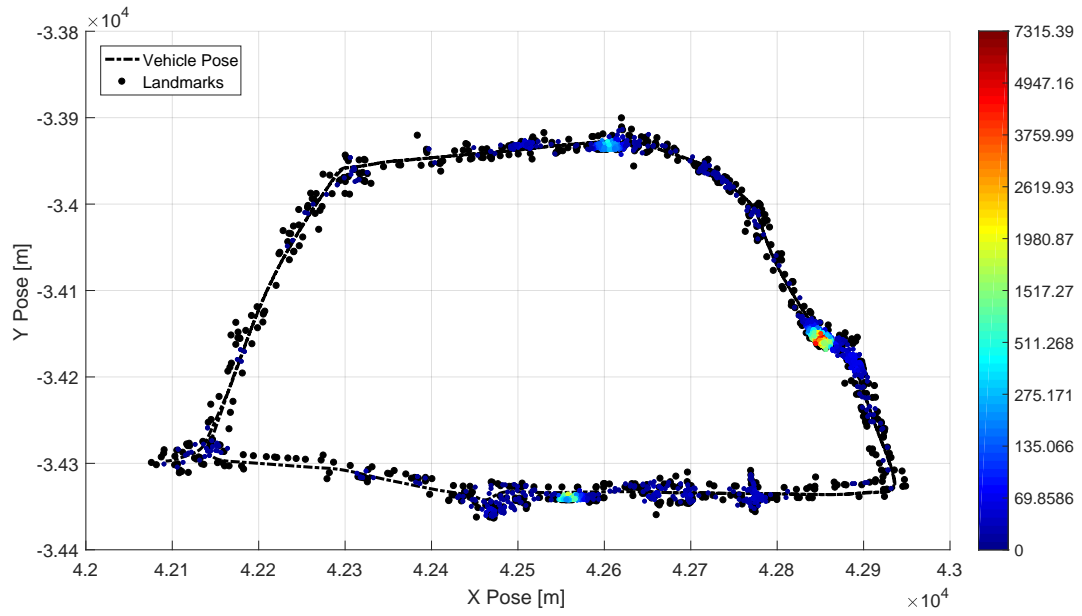


Figure 6.35: SIFI5: Heatmap representing ambiguous constellation density (by centroid) over mapped area

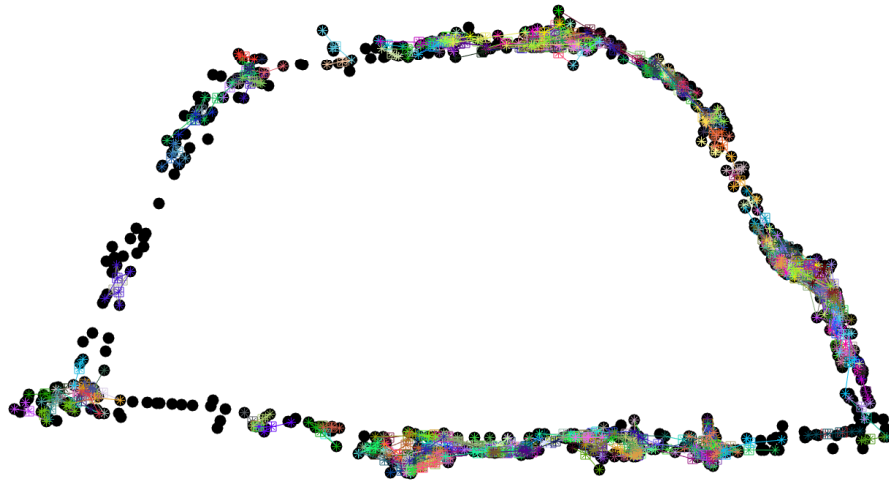


Figure 6.36: SIFI5: View all ambiguous constellations of 3 vertices

Along the same lines as Figures 6.15-6.16, Figures 6.37-6.38 illustrate portions of a vehicle trajectory where the translational and rotational errors could be large. The segments of the trajectory that are red indicate regions of high error should sufficient incorrect associations be chosen. Likewise, blue regions indicate regions of low error. When comparing Figure 6.35 to the position solution of vehicle in Figure 6.23, regions with dense ambiguous constellation centroids correlate with those where the solution becomes oscillatory.

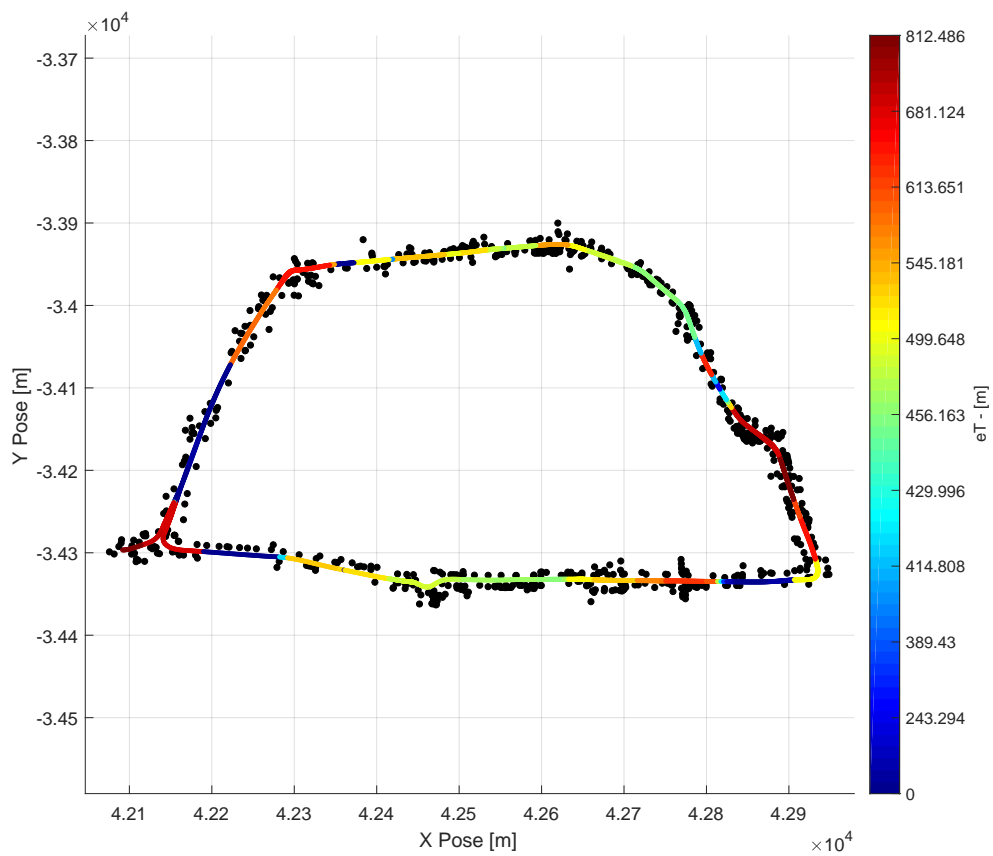


Figure 6.37: SIFI5: Heatmap representing translational error from ambiguous features over mapped area

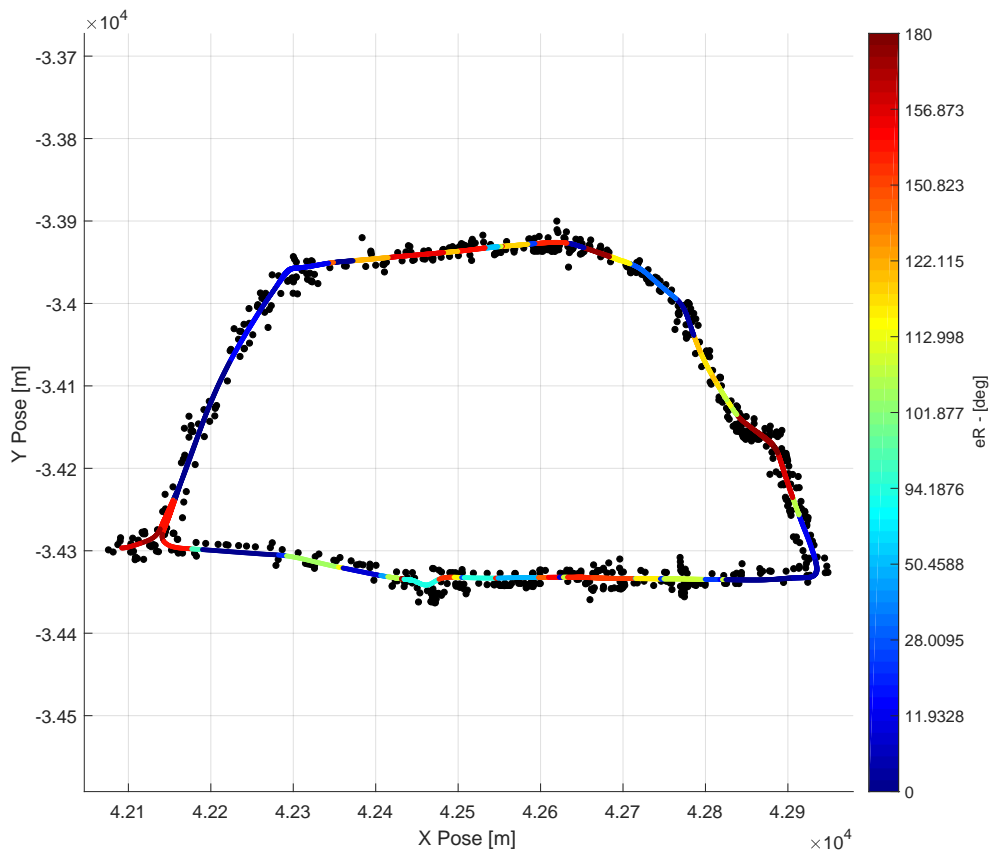


Figure 6.38: SIFI5: Heatmap representing rotational error from ambiguous features over mapped area

One metric that does align with what was discovered in the simulation is the association percentage between the fine and coarse data sets which can be seen in Tables 6.14-6.15. It's important to distinguish the percent associations and verifications in these tables from those presented in the Section 6.1.3. Previously, the percentages were reported in a cascaded manner where the percent associations only accounts for observations where detections were recorded. In this section, the percent associations is reported based on the total amount of observations in the sample. This explains why the numbers are drastically different between the two sets of results.

As explained above, the coarse resolution hash table has larger bins which can help account for noise or other boundary condition scenarios. This gives the coarse data set a slight advantage over the fine resolution data when finding associations at each recognition step. On

the negative side, it also opens the possibility for more incorrect associations as well, which is shown because both coarse and fine solutions share about the same verified association percentages. This maintains that the verification step is necessary, because it eliminates a significant amount of potential misleading information. Not everything is filtered correctly however, as there are cases where misleading information lead to a significant discontinuity in the vehicle position or heading. One particular instance of this occurs around the 60-100 second mark as shown in Figure 6.31 and the accompanying plots (Figures 6.28, 6.29, 6.30). In the latter grouping of figures, it can be seen that green dots are very scattered and do not overlap with the reference solution as they are expected to be.

Table 6.14: SIFI5: Result statistics

Metric	Value
Percent Converged	63.8525
Percent Converged X	71.4235
Percent Converged Y	82.1273
Percent Converged Yaw	94.8820
Percent Detections	94.5981
Percent Associations	24.1342
Percent VerifiedAssociations	20.3442
Average Detections	11.4241
Average Associations	15.9946
Average VerifiedAssociations	16.7784
Average ProcessingTime [ms]	43.4925

Figure 6.39 shows an example of how this occurs from the visualization where red circles and lines indicate mapped features, blue circles and lines represent detections, and white lines depict the association chosen. Comparing these same regions with those from the fine data set in Figures 6.20, 6.21, 6.22, 6.23 show much higher fidelity when traversing through these regions.

Table 6.15: SIFI20: Result statistics

Metric	Value
Percent Converged	48.2432
Percent Converged X	55.2049
Percent Converged Y	70.2347
Percent Converged Yaw	81.1220
Percent Detections	86.8044
Percent Associations	47.9455
Percent VerifiedAssociations	22.8223
Average Detections	7.1071
Average Associations	5.9265
Average VerifiedAssociations	7.0134
Average ProcessingTime [ms]	70.0965

It can also be shown that the coarse resolution creates a significantly larger amount of data to search through at each update. On average, a recognition step takes twice the amount of processing time when compared to the fine data set. This is also directly correlated to the average number of features detected at each update. One final item to address, is the large consecutive instances of data with detects and no associations. In these areas, the number of detections is very close to the minimally acceptable value required for an update. If there is no strong evidence that any particular candidate feature set is likely over another, the recognition phase will simply abandon the entire association step and provide no matches. Additionally, as can be shown from the topics in Chapter 3, there are larger amounts of ambiguous constellations as the number of features decreases. In other words, more features will help disprove ambiguities and improve uniqueness.

6.2.2 Karlsruhe

For each data set collected on the Karlsruhe circuit, the hash tables were generated using a map of 891 cylindrical features assumed to be perturbed by Gaussian White Noise of a variance

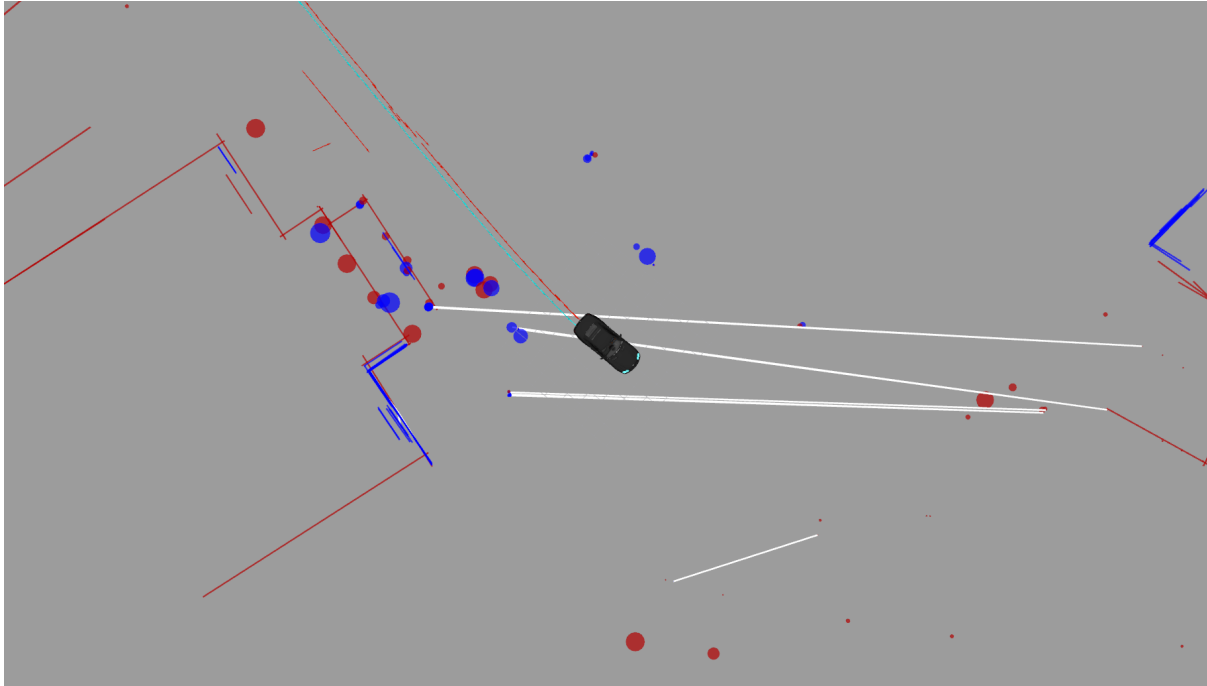


Figure 6.39: Incorrect feature association caused by ambiguous geometry

of 25cm^2 . The perturbation was assumed equal in both the lateral and longitudinal map axes. In addition to the hash values representing 2D positions, the recognition step utilized the radius of the cylinders as an additional matching descriptor.

6.2.2.1 Fine Resolution – KIT5

For the high resolution data set on the Karlsruhe circuit, the hash table was discretized using a quantization parameter q_{pose} of 5 centimeters and collisions were dealt with using the strict filter. Additionally, no bases were created from features separated further than 60 meters (b_{limit}) and all features outside a 100 meter radius from each basis origin (r_{incl}) were excluded from the layer data. Additional runtime parameters for the fine resolution run can be seen in Table 6.16. Features were matched using Algorithm 4.7 as presented in Chapter 4. The hash table produced from these parameters contained 1,965,610 hash entries and comprised of 30,712 layers resulting in a file size of 324 MB. In the following, this scenario will be identified as KIT5.

Figure 6.40 shows the path of the vehicle along with all the landmarks extracted from the circuit presented previously in Figure 5.6. A breakdown of the solution state at each update is

Table 6.16: KIT5: Parameters

Metric	Value
q_{pose} [m]	0.05
q_{radius} [m]	0.05
b_{limit} [m]	60.00
r_{incl} [m]	100.00
s_{trans}	1.00
Collision Filtering	strict
Descriptors	none

also depicted to indicate areas where the solution was unable to perform as expected. As the legend explains, pink "X" symbols indicate sample updates where no landmark detections were found. Yellow triangles show updates where landmarks were detected, but insufficient associations were found to utilize the data. Blue squares indicate areas where matches were found, but did not pass the verification procedure whereas green circles met all necessary criteria to report a positioning correction to the solution. Figures 6.41, 6.42, and 6.43 all utilize the same legend, but show the positioning solution as an individual dimension over time.



Figure 6.40: KIT5: Overlay of odometry with map of landmarks

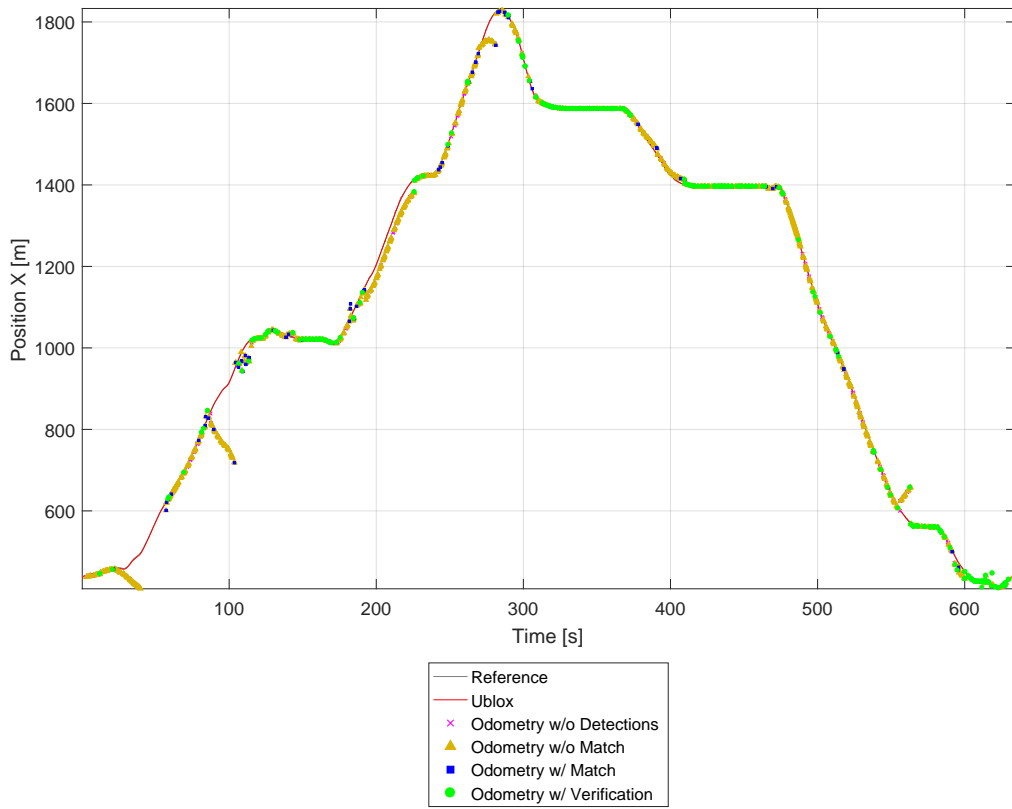


Figure 6.41: KIT5: Breakdown of X position with update availability

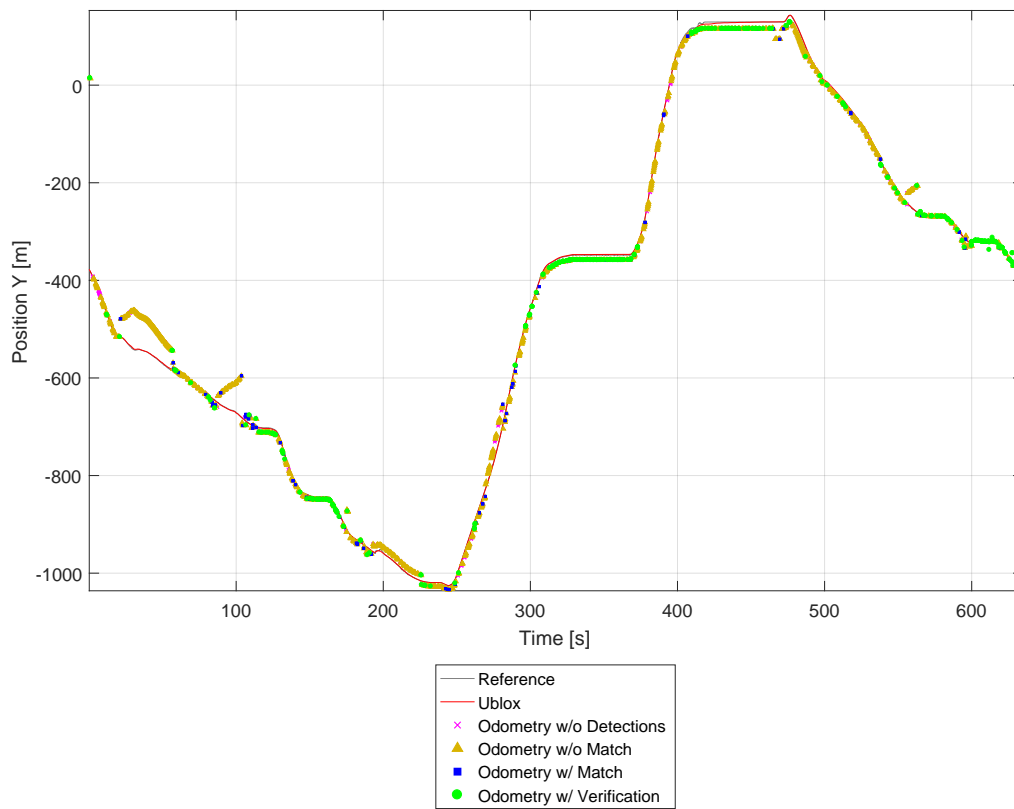


Figure 6.42: KIT5: Breakdown of Y position with update availability

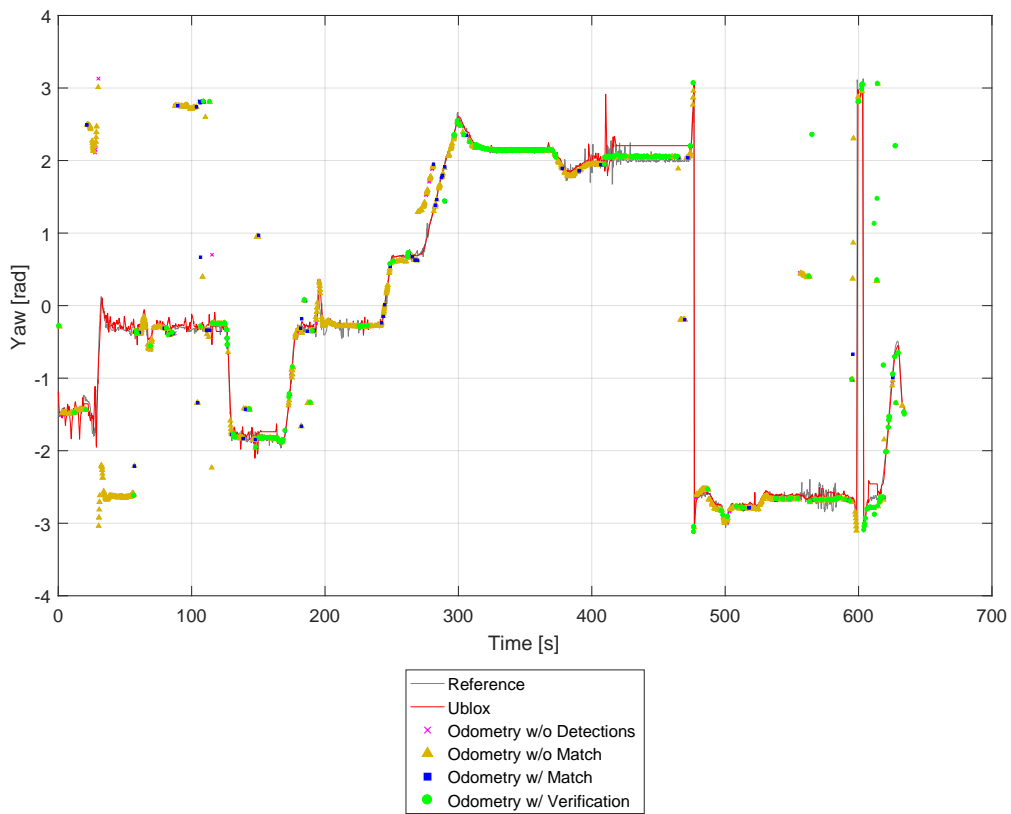


Figure 6.43: KIT5: Breakdown of heading with update availability

Figure 6.44 shows the horizontal positioning and heading solutions from the data set. As mentioned above, the reference solution is the result of the algorithm presented in Section 5.2. The data marked with "Ublox" is a Global Navigation Satellite Systems (GNSS) solution fused with Inertial Navigation Systems (INS) solution only. Lastly, the data marked "Odometry" is the Geometric Hashing Localization solution.

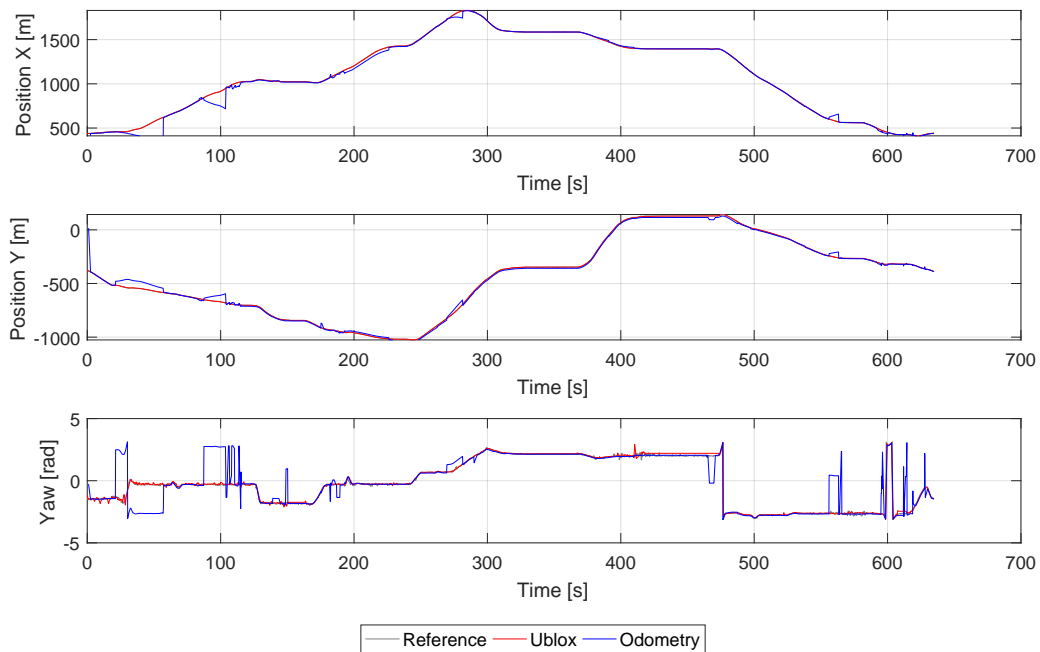


Figure 6.44: KIT5: Planar position and heading of the vehicle over time

Figure 6.45 depicts the error between the Geometric Hashing Localization solution and the reference solution and is segregated according to the convergence rules presented in Section 6.2 where red indicates the raw error and blue only represents a converged result.

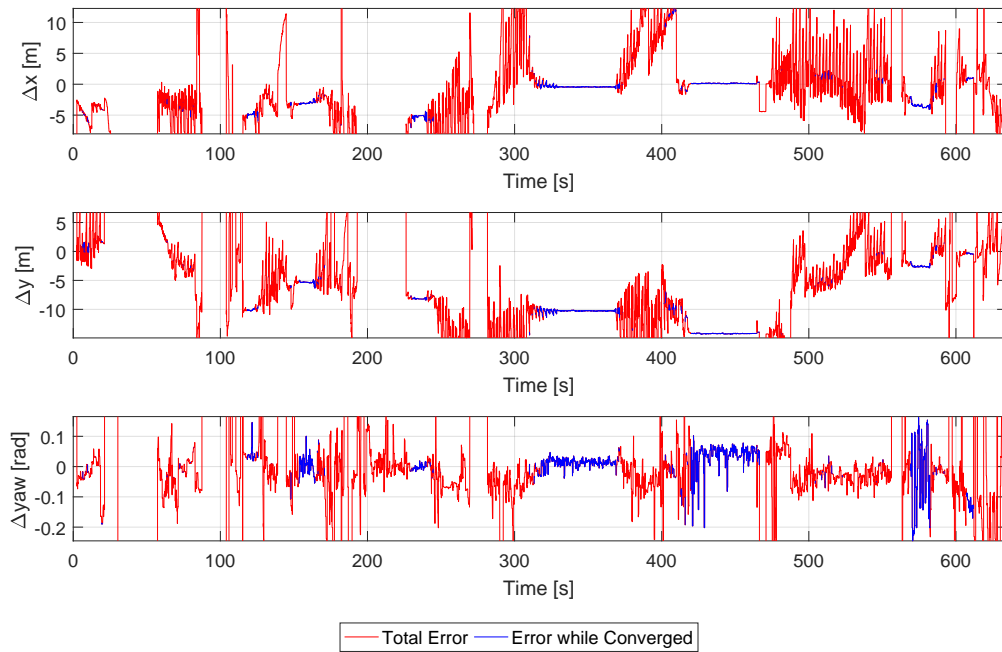


Figure 6.45: KIT5: Error between reference solution and odometry

Figure 6.46 shows association results of the solution. Black lines indicate the total number of detected features for the given sample. Red and green lines each indicate the number of associations for the sample. In areas where black is clearly visible, there exist no associations. Green represents areas where associations were considered valid whereas red lines failed the matching process at some stage. Lastly, Figure 6.47 depicts a relationship between the number of extracted features and the total processing time of the update.

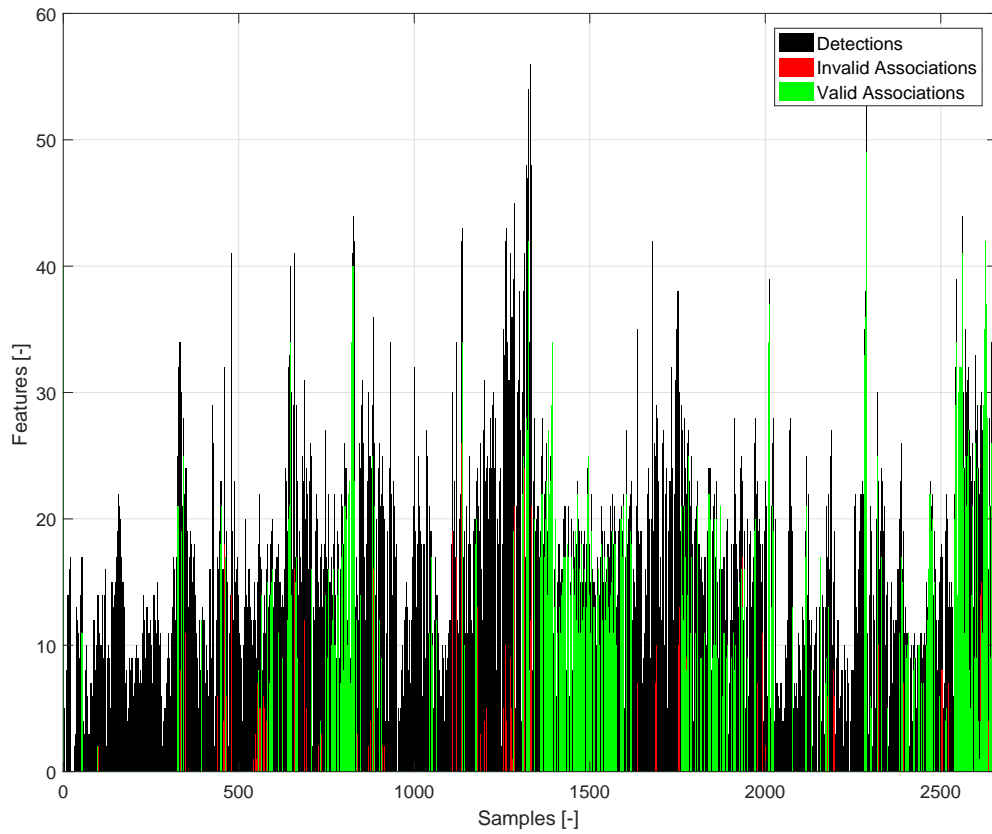


Figure 6.46: KIT5: Measurement correction availability during the drive

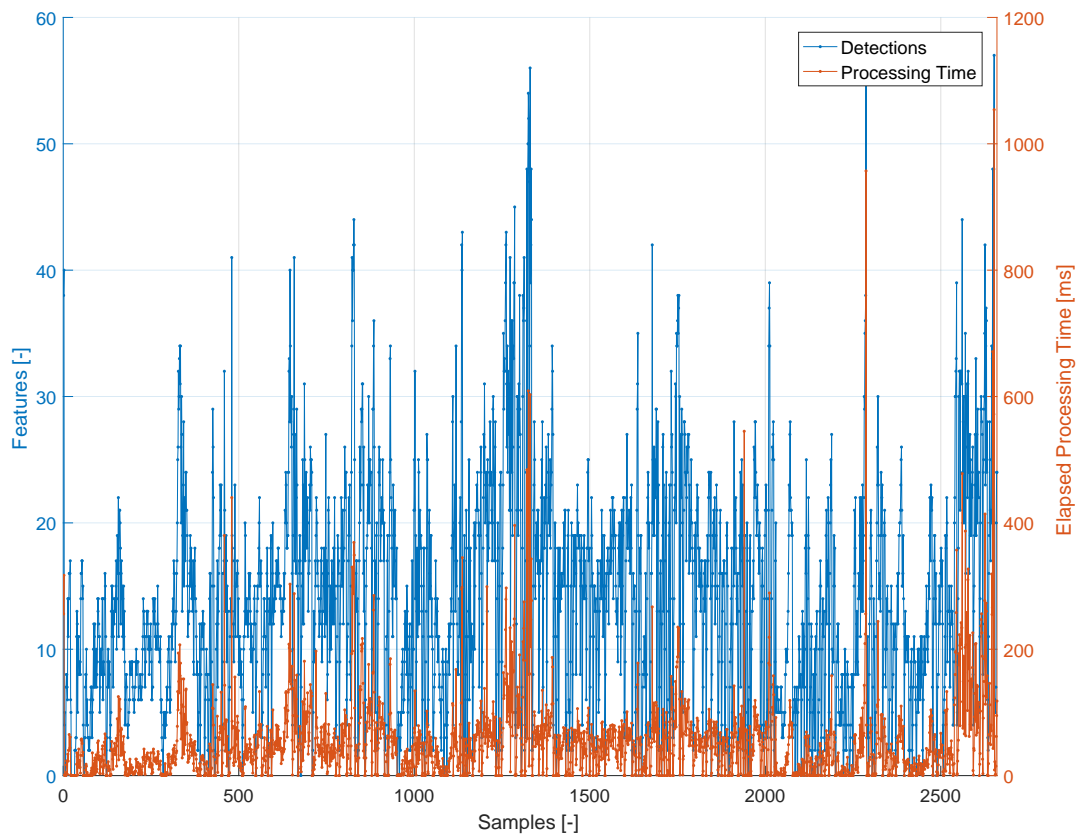


Figure 6.47: KIT5: Processing time for each detection during the drive

6.2.2.2 Course Resolution – KIT20

For the course resolution data set on the Karlsruhe circuit, the hash table was discretized using a quantization parameter q_{pose} of 20 centimeters and collisions were also dealt with using the strict filter. Additionally, no bases were created from features separated further than 60 meters (b_{limit}) and all features outside a 100 meter radius from each basis origin (r_{incl}) were excluded from the layer data. Additional runtime parameters for the coarse resolution run can be seen in Table 6.17. Again, features were matched using Algorithm 4.7 as presented in Chapter 4. The hash table produced from these parameters contained 636,606 hash entries and comprised of 30,712 layers resulting in a file size of 184 MB. In the following, this scenario will be identified as KIT20.

Table 6.17: KIT20: Parameters

Metric	Value
q_{pose} [m]	0.20
q_{radius} [m]	0.05
b_{limit} [m]	60.00
r_{incl} [m]	100.00
s_{trans}	1.00
Collision Filtering	strict
Descriptors	none

Figure 6.48 again shows the path of the vehicle along with all the landmarks extracted from the circuit presented in Figure 5.6. Figures 6.49, 6.50, and 6.51 again show the individual positions and all utilize the same legend as was previously stated.

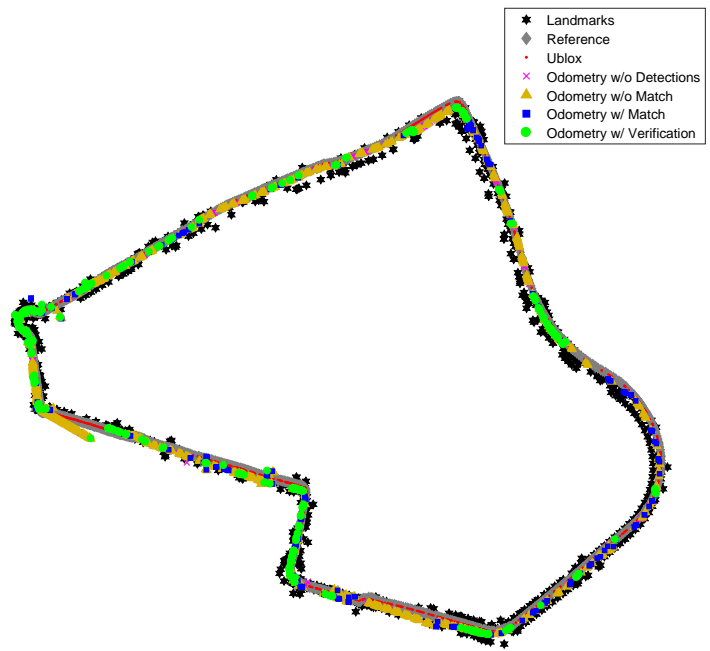


Figure 6.48: KIT20: Overlay of odometry with map of landmarks

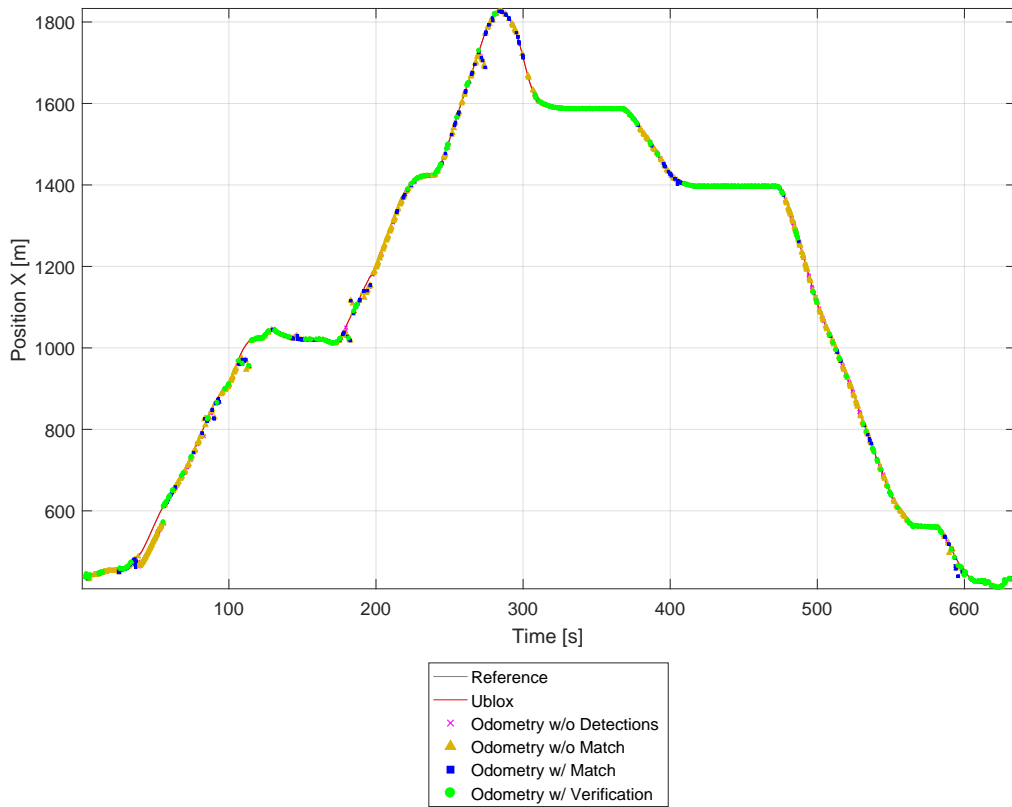


Figure 6.49: KIT20: Breakdown of X position with update availability

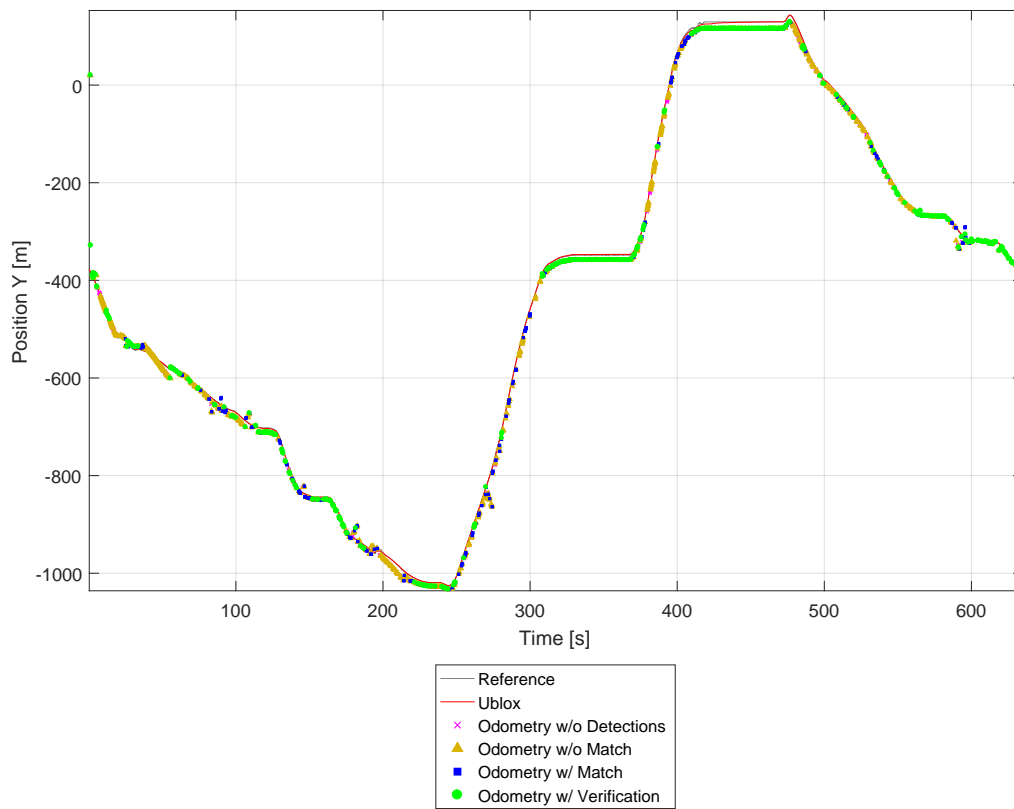


Figure 6.50: KIT20: Breakdown of Y position with update availability

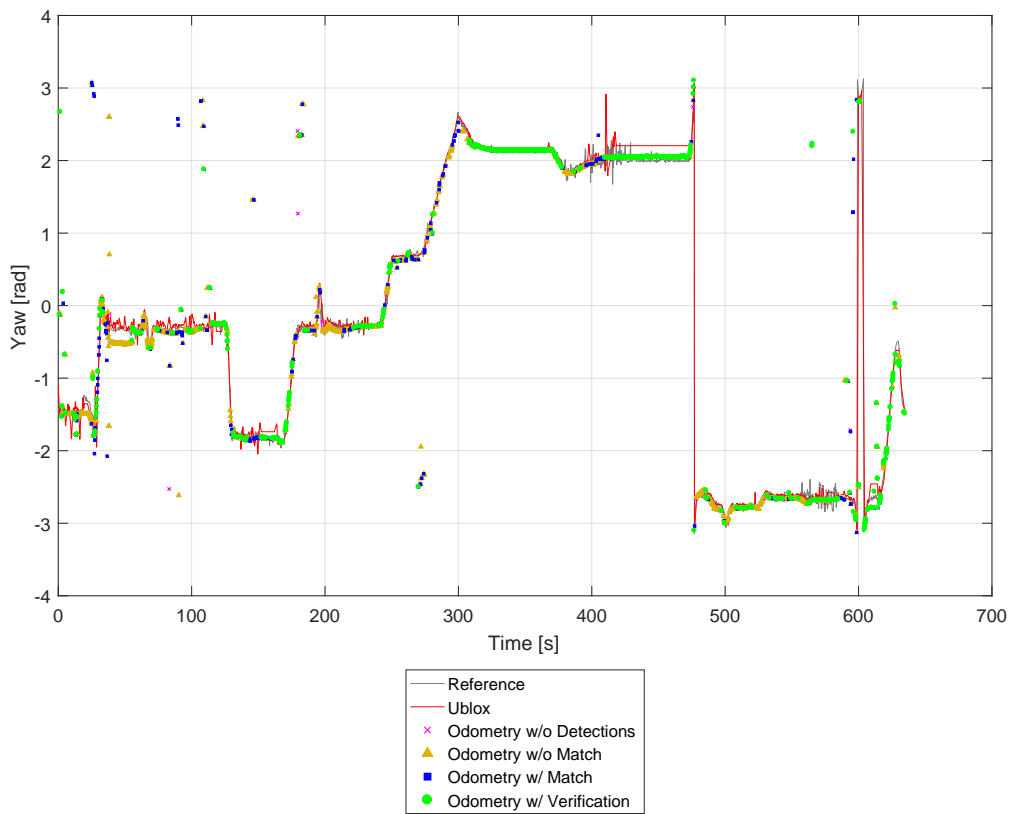


Figure 6.51: KIT20: Breakdown of heading with update availability

Similar again to 6.44, Figure 6.52 shows the three positioning solutions from the data set and Figure 6.53 depicts the error between the Geometric Hashing Localization solution and the reference solution. Red lines report the raw positioning solution while blue lines indicate periods of convergence according to the rules presented in Section 6.2.

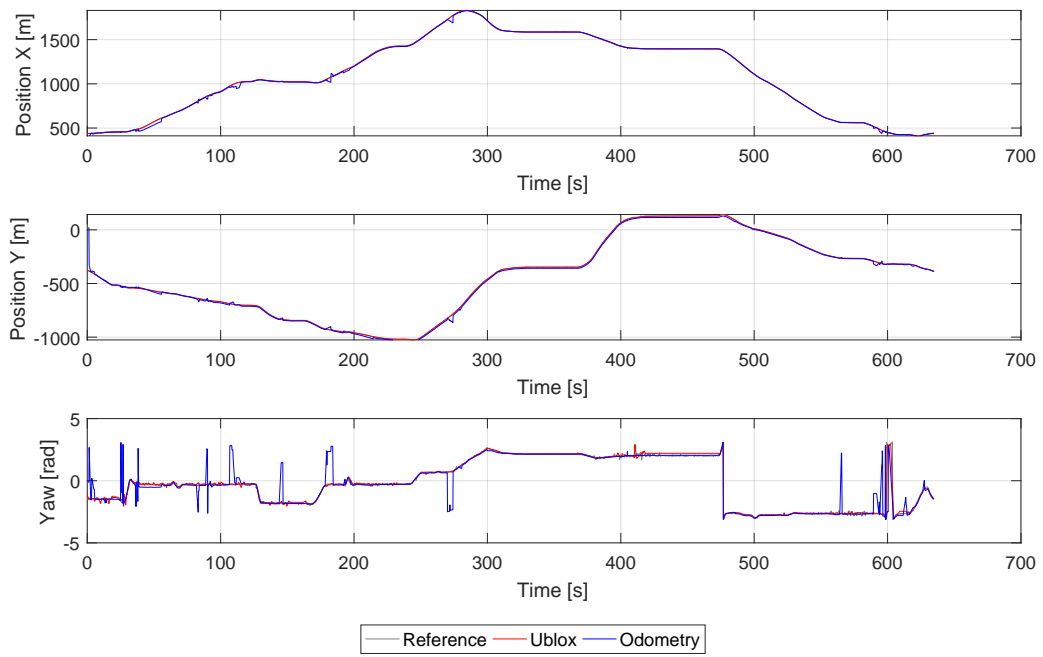


Figure 6.52: KIT20: Planar position and heading of the vehicle over time

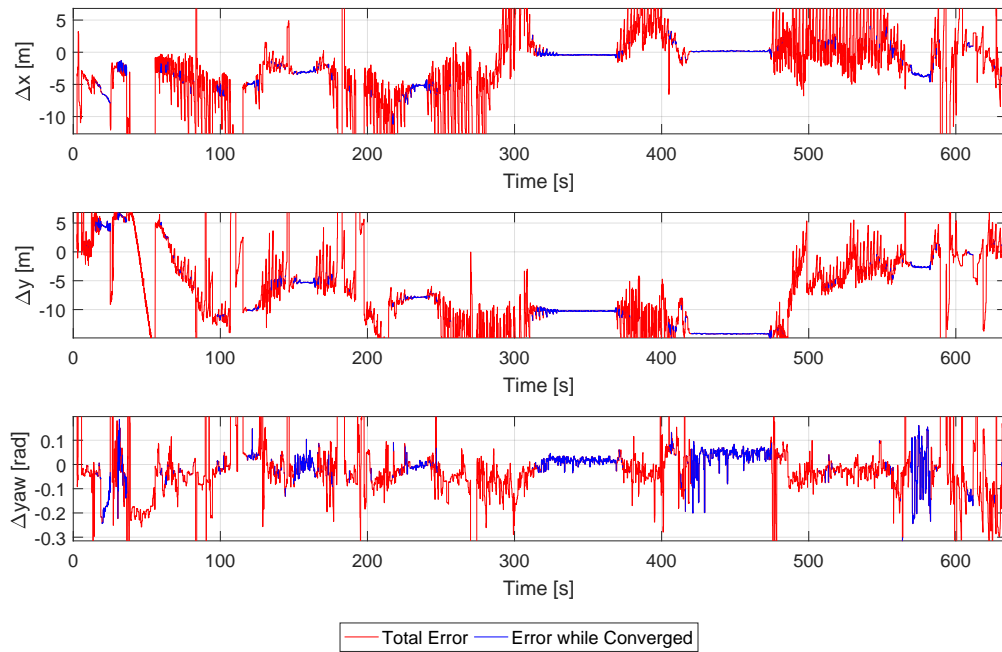


Figure 6.53: KIT20: Error between reference solution and odometry

Figure 6.54 shows association results of the solution in the same format that was presented in 6.46. Finally, Figure 6.55 again depicts a relationship between the number of extracted features and the total processing time of the update.

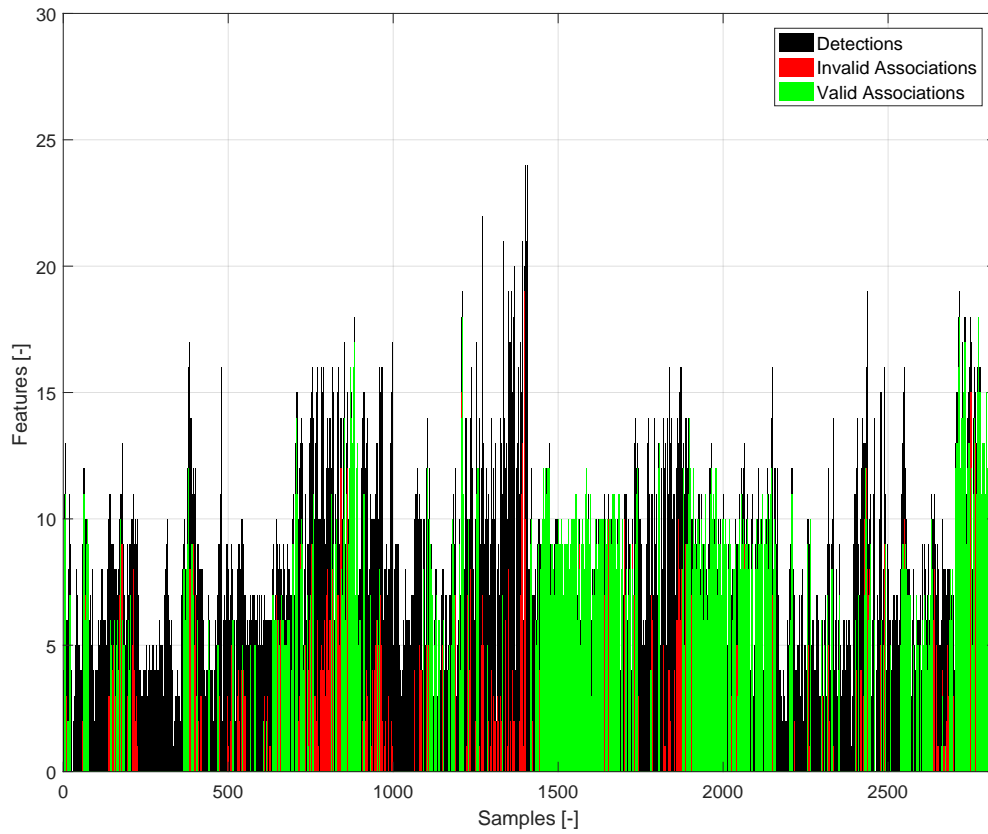


Figure 6.54: KIT20: Measurement correction availability during the drive

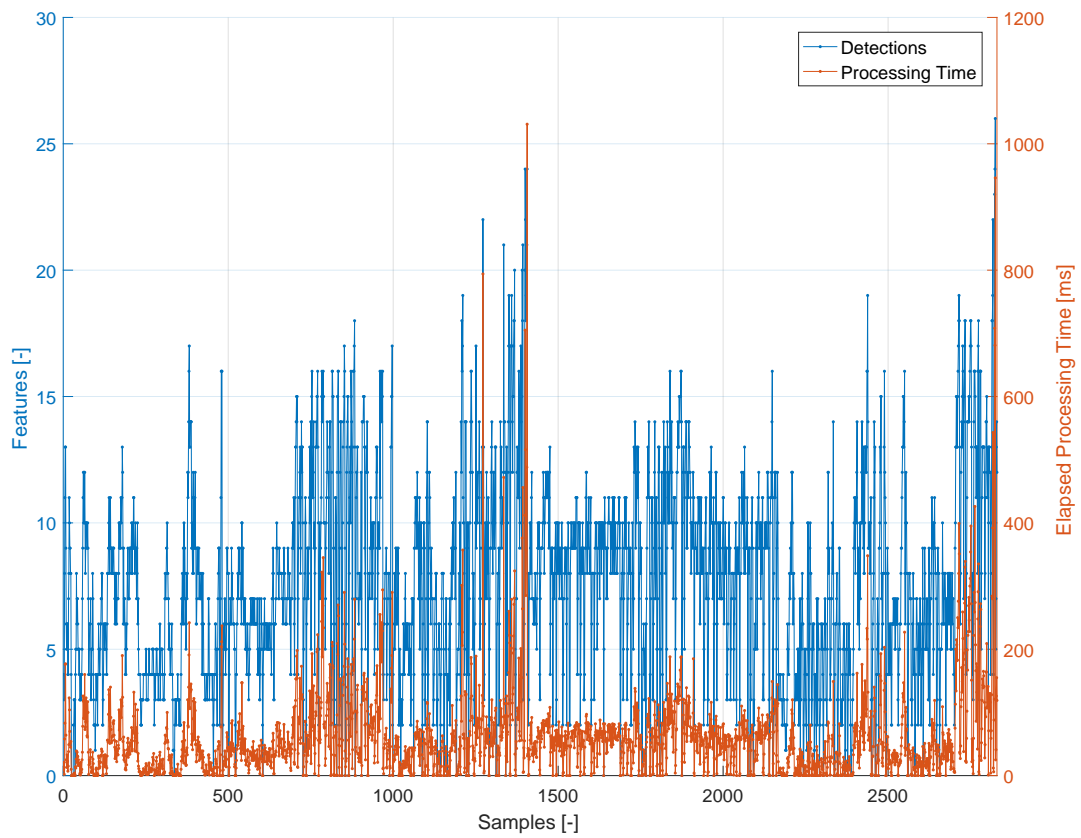


Figure 6.55: KIT20: Processing time for each detection during the drive

6.2.2.3 Discussion of Karlsruhe Results

The results from the Karlsruhe data set shown in Tables 6.18-6.19 do not closely follow the same trends established from the simulation and Sindelfingen data sets. Notably, as shown in Tables 6.20-6.21, there is not a huge difference in performance between the coarse and fine scenarios even though the coarse solution spent more time converged and had a larger percent of valid associations. Instead, the differences can be seen when looking at the total RMS error (*i.e.* converged and non-converged together) where the coarse solution performed noticeably better. This is an indication that the effects (both positive and negative) presented by coarse and fine scenarios fell beneath the convergence floor.

Table 6.18: KIT5: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Ublox	2.9070	3.0924	0.2911
Odometry	51.7512	31.9894	1.0653
Odom Converged	2.8817	10.1477	0.0560

Table 6.19: KIT20: Error metrics

RMS Error	x [m]	y [m]	yaw [rad]
Ublox	3.0861	2.9169	0.3249
Odometry	22.1847	22.2974	0.6978
Odom Converged	2.9820	9.9705	0.0647

In general, the Karlsruhe environment poses a more difficult challenge for the recognition phase due to the large amounts of symmetry and feature clustering. Some areas in the map contain up to 55 features within the FoV whereas there are long periods within highly symmetric areas containing only 5 or 6 features. These challenges contribute heavily to the lower convergence percentages across the fine and the coarse scenarios when compared to the Sindelfingen data sets.

Table 6.20: KIT5: Result statistics

Metric	Value
Percent Converged	27.8404
Percent Converged X	39.7233
Percent Converged Y	41.6692
Percent Converged Yaw	75.0083
Percent Detections	91.6165
Percent Associations	23.0075
Percent VerifiedAssociations	18.3083
Average Detections	15.3578
Average Associations	15.1471
Average VerifiedAssociations	16.7926
Average ProcessingTime [ms]	52.9466

Two particularly troublesome areas occur at the 30 and 85 second marks respectively where an instance of an ambiguous constellation guides the localization solution off track. This can easily be seen when comparing Figure 6.44 and Figure 6.56. To accompany this, Figure 6.57 shows all the locations of ambiguous constellations similar to Figure 6.36 presented previously. Figures 6.58-6.59 show the high risk trajectory segments based on the maximal translational and rotational error of ambiguous constellation occurrences.

Table 6.21: KIT20: Result statistics

Metric	Value
Percent Converged	35.0347
Percent Converged X	47.3573
Percent Converged Y	45.9127
Percent Converged Yaw	87.2631
Percent Detections	91.3781
Percent Associations	49.6466
Percent VerifiedAssociations	31.6254
Average Detections	8.3569
Average Associations	7.1786
Average VerifiedAssociations	8.5039
Average ProcessingTime [ms]	59.4042

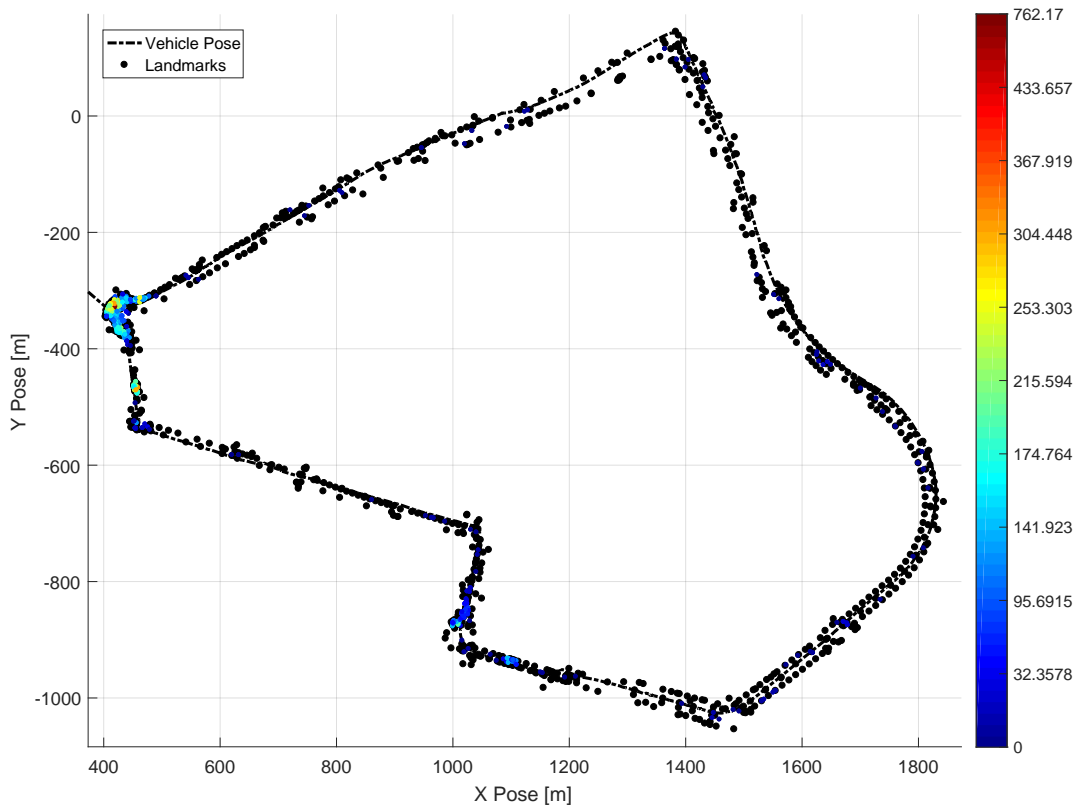


Figure 6.56: KIT5: Heatmap representing ambiguous constellation density (by centroid) over mapped area

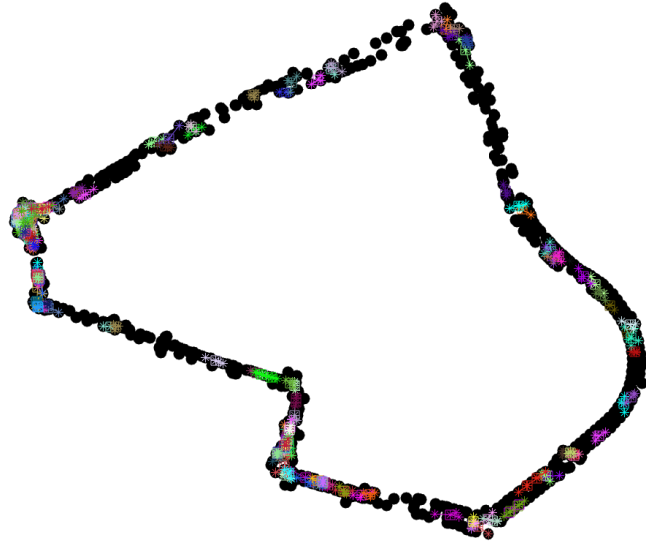


Figure 6.57: KIT5: View all ambiguous constellations of 3 vertices

In each case where the solution jumps off track, the set of samples following these errant associations contain very few features. This reduces the system's ability to correct the erroneous positioning and continue down the wrong path. The coarse solution fairs significantly better in these areas. This is largely due to the quantization boundary condition issues presented earlier in Chapter 4. Another notable situation occurs around the 400 second mark in the fine resolution data set. As can be seen in the lower left corner of Figure 6.40, the odometry solution appears to turn too early. Several yellow dots, indicating no associations, are trailed by a blue dot and finally a green dot which snaps the positioning solution back on the correct track. This depicts a gradual process of the solution regaining a converged position.

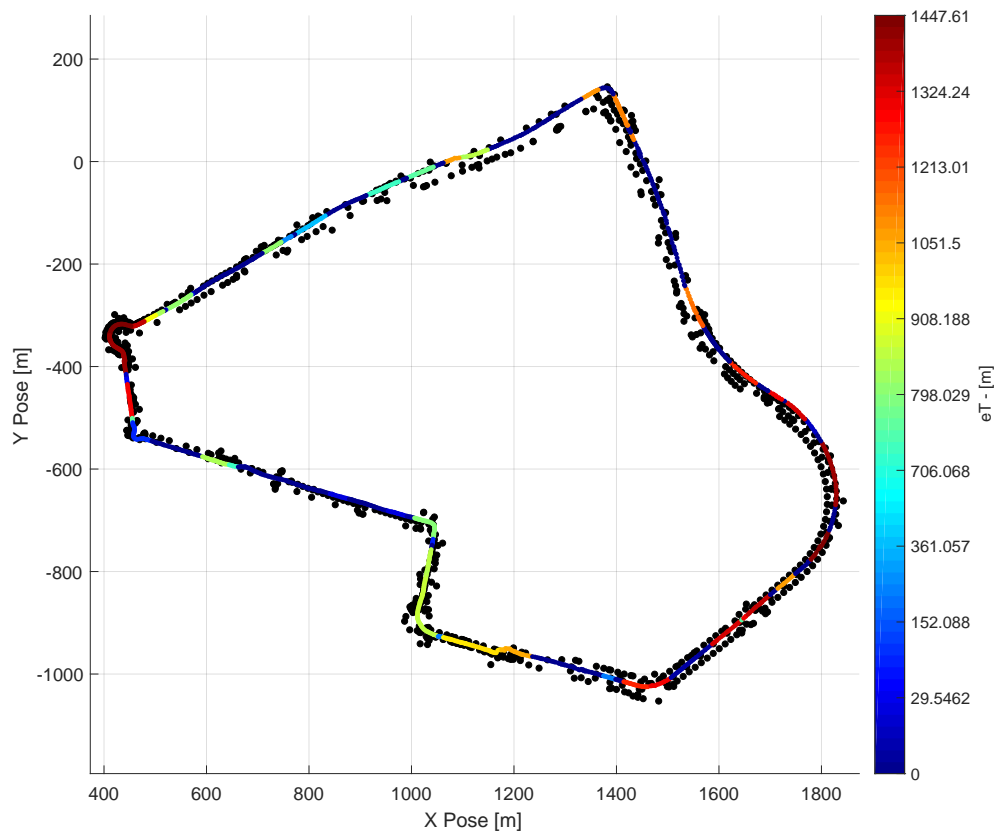


Figure 6.58: KIT5: Heatmap representing translational error from ambiguous features over mapped area

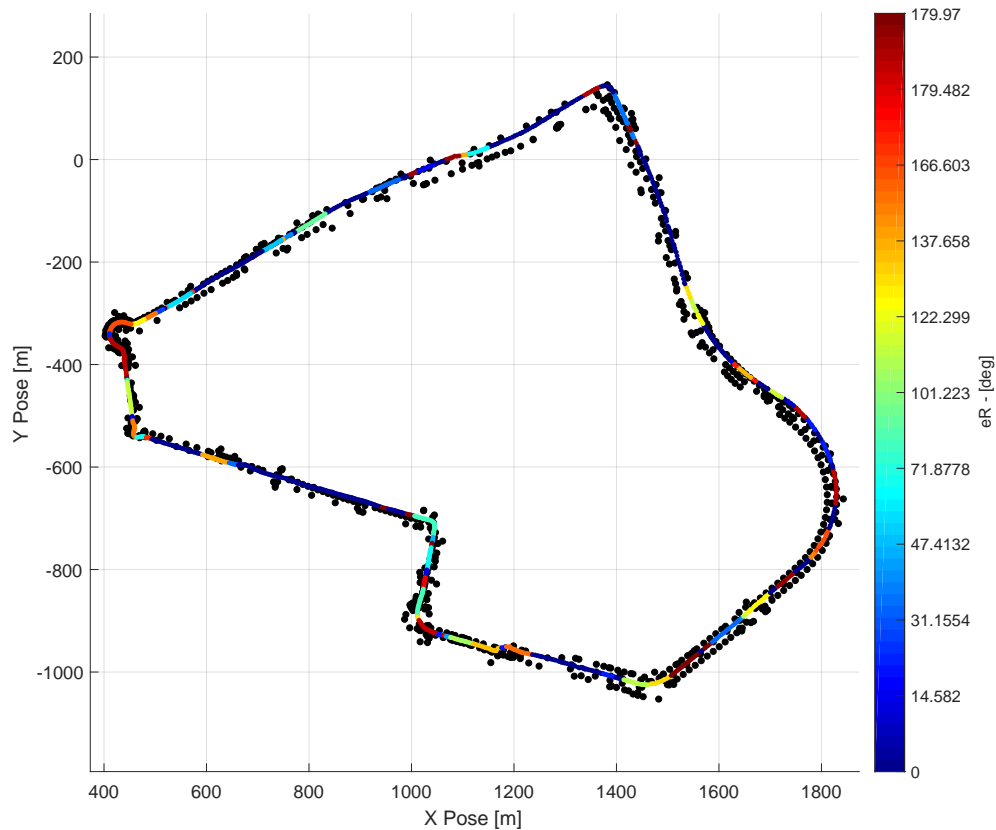


Figure 6.59: KIT5: Heatmap representing rotational error from ambiguous features over mapped area

Overall, the Karlsruhe results performed worse in comparison to the Sindelfingen data sets. This may be an indication of a misalignment between the reference and Geometric Hashing solutions or other errors or miscalibration in the feature extraction steps. This theory is also reinforced by comparing the ambiguous geometry results between Sindelfingen and Karlsruhe. It can be seen that the maximal error values from Figures 6.58-6.59 are much lower than Figures 6.37-6.38. In addition, the ambiguous constellation centroid density is much smaller in Figure 6.56 than in Figure 6.35.

Chapter 7

Conclusion and Future Work

In this thesis, a novel approach to laser-based feature association using Geometric Hashing was presented within the context of ground vehicle localization. The Geometric Hashing algorithm was re-introduced and discussed for working with noisy features under isometric transformation constraints. A Bayesian approach to characterizing noise on extracted features during matching was proposed to over-bound and capture the true covariance ellipse for isometric invariants. In addition, a new approach to understanding and utilizing map symmetry using the concepts of Geometric Hashing was presented. This approach enables the recognition phase to obtain a maximum bound on possible error should the incorrect association be chosen when conflicted with an ambiguity. It also yields information towards a real-time integrity monitor to verify no hazardous misleading information is fed into the system.

Several different matching methods were proposed and tested for use during the recognition phase. Most of the schemas centered around two common themes: matching based on a basis or matching based on individual features. When matching bases, it was found that there was an increased likelihood of obtaining correct associations, but a decreased robustness to noise and overall provided less frequent useful updates to the system. On the other hand, matching features allowed for using partial information across several bases, but increased possibility of incorrect associations.

7.1 Conclusions on Geometric Hashing

The approach of using Geometric Hashing for ground vehicle localization was shown to be feasible via simulation and integration with real system data. Results show that localization within a mapped region using Geometric Hashing Localization is possible and can compete with GPS precision but fell short on requirements for availability and integrity. To obtain

these results, the implementation was first tested in a simulation environment with ideal conditions. The outcome proved that given no feature ambiguity and a sufficient amount of extracted features, geometric hashing can correctly associate features with previously mapped counterparts leading to a highly precise positional state correction. Next, the algorithm was integrated into an existing feature-based localization framework to provide the data association component and understand how the performance changed under more realistic conditions. Data such as extracted feature measurements, GNSS positions, and other vehicle parameters were pre-collected from driving around two different circuits in Baden-Württemberg, Germany. These measurements were played back onto a computer running the full localization pipeline to emulate a real drive. The outcome from these tests showed that Geometric Hashing Localization is capable of delivering high precision localization updates in a timely manner, but under performs in some key areas preventing effective use as a primary localization solution for safety critical applications.

One shortcoming of the data from the integrated localization solution was the sampling rate of data. Each sample from the results of the two circuits was recorded at the end of a localization update rather than at a fixed, asynchronous rate. This means that for updates that require more processing time, the system aggregated (or in some cases dropped) information that could have been used in a subsequent update. Thus, there is a correlation between the processing time of one update, and the ability to find associations in the next. The GNSS solution was also sampled at this same frequency. This gives explanation as to why there was a large difference in GNSS RMS error and percent of detected features between the coarse and fine solutions of the same route.

Interestingly, the algorithm maintained precise vehicle heading in comparison to horizontal positioning. Even in areas where the localization solution tended to diverge from the path, the system still maintained an accurate estimate of heading in many cases. More study is required to understand if this is universally true for all scenarios or if this is just an artifact of the data studied in this thesis.

7.2 Future Work

Overall, Geometric Hashing Localization delivered on many internally set requirements. It was demonstrated that feature associations are time-invariant and can meet the needs of a real-time system. Although not explicitly tested in this thesis, the system can be extended to function using semantically chosen features from both LiDAR and imagery within both indoor and outdoor environments. The approach can also scale up to larger map regions and can be parallelized to further improving timing and ambiguity rejection.

However, there is still plenty of improvements that can also be made to the approach. Even though results show promise for highly precise positioning updates, the availability of these updates is below expectations for safety critical use-cases. Largely, this issue is a result of two different problems; lack of uniform feature density within the mapped region and ambiguous or noisy extracted features.

7.2.1 Additional Primitives and Descriptors

One method to help improve the likelihood of each update containing features is to add more primitives to the feature set. This could also improve feature density per update as well. Within this work, only cylindrical features were considered for hashing, but other features (such as planes, curbs, and road signs) could be included as well. Planes in particular can be extremely beneficial to positioning, but suffer greatly when partially occluded. If used in conjunction with a more reliable primitive like cylinders or road signs, some of this information could be partially recovered.

One major way to eliminate some ambiguity that was not implemented in this work is to utilize more information for each primitive. Cylinders can be described simply by their position in a 2D plane, but can also have additional characteristics such as radius, height, and tilt and azimuth in 3D cases. Including more detail from each cylinder could improve the matching process by adding additional characteristics to match against.

7.2.2 Integrity Risk Monitor

Another major improvement that could be made to improve the existing implementation is obtaining a true characterization of the noise features undergoing rigid transformation. Currently, as was presented in Section 4.2.1.1, the joint probability density function for features undergoing similarity transformations is known. However, when applying a similar approach to the rigid noise model, the joint probability density function is no longer Gaussian in nature. An example of this is shown in *Hofstetter et al* [26] by creating a small monte carlo simulation on rigidly transformed noisy features. Plotting the results of the monte carlo, it was seen that the points spread over a "curved slot" shape. In this thesis, this curved slot was over-bounded with an ellipse, but this does not truly characterize the noise and will negatively affect the matching procedure.

Secondly, more steps could be taken to yield better information for an online integrity monitor. This thesis laid the groundwork for deterministically computing all instances of ambiguous map features and storing them in an index friendly manner. Next, a method to utilize this information to compute the probability of a correct or incorrect association at each update should be developed. With these two probabilities considered, the total probability of hazardous misleading information is known and can be used to monitor alert limits for safety critical applications as described in *Joerger's* work [2].

References

- [1] Ömer Şahin Taş, Niels Ole Salscheider, Fabian Poggenhans, Sascha Wirges, Claudio Bandera, Marc René Zofka, Tobias Strauss, J. Marius Zöllner, and Christoph Stiller. Making bertha cooperate—team annieway’s entry to the 2016 grand cooperative driving challenge. *IEEE Transactions on Intelligent Transportation Systems*, 19(4):1262–1276, 2018.
- [2] Mathieu Joerger, Michael Jamoom, Matthew Spenko, and Boris Pervan. Integrity of laser-based feature extraction and data association. In *2016 IEEE/ION Position, Location, and Navigation Symposium (PLANS)*, pages 557–571. IEEE/ION, 2016.
- [3] F. Schuster, C. G. Keller, M. Rapp, M. Haueis, and C. Curio. Landmark based radar slam using graph optimization. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2559–2564, 2016.
- [4] Ziyang Hong, Yvan Petillot, and Sen Wang. Radarslam: Radar based large-scale slam in all weathers. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5164–5170, 2020.
- [5] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1985–1991. IEEE, 2003.
- [6] Cyrill Stachniss Giorgio Grisetti and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *2005 International Conference on Robotics and Automation (ICRA)*, pages 2443–2448. IEEE, 2005.
- [7] Isidore Rigoutsos. *Massively parallel Bayesian object recognition*. PhD thesis, New York University, Department of Computer Science, 1992.

- [8] Yehezkel Lamdan, Jacob T Schwartz, and Haim J Wolfson. Object recognition by affine invariant matching. In *Proceedings CVPR'88: The Computer Society Conference on Computer Vision and Pattern Recognition*, pages 335–344. IEEE, 1988.
- [9] Yehezkel Lamdan, JT Schwatrtz, and Haim J Wolfson. On recognition of 3-d objects from 2-d images. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1407–1413. IEEE, 1988.
- [10] Haim J Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE computational science and engineering*, 4(4):10–21, 1997.
- [11] Jacob T Schwartz and Micha Sharir. Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves. *The International Journal of Robotics Research*, 6(2):29–44, 1987.
- [12] Yehezkel Lamdan and Haim J Wolfson. Geometric hashing: A general and efficient model-based recognition scheme, 1988.
- [13] W Eric L Grimson and Daniel P Huttenlocher. On the sensitivity of geometric hashing. In *[1990] Proceedings Third International Conference on Computer Vision*, pages 334–338. IEEE, 1990.
- [14] Yehezkel Lamdan and Haim J Wolfson. On the error analysis of 'geometric hashing'. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 22–27. IEEE, 1991.
- [15] Frank CD Tsai. A probabilistic approach to geometric hashing using line features. *Computer Vision and Image Understanding*, 63(1):182–195, 1996.
- [16] Andrea Califano and Rakesh Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):373–392, 1994.

- [17] Andrew Gilbert and Richard Bowden. Geometric mining: Scaling geometric hashing to large datasets. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 16–24, 2015.
- [18] Masahiro Tomono. A scan matching method using euclidean invariant signature for global localization and map building. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 1, pages 866–871. IEEE, 2004.
- [19] Mauro S Costa, Robert M Haralick, and Linda G Shapiro. Optimal affine-invariant point matching. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume 1, pages 233–236. IEEE, 1990.
- [20] Isabell Hofstetter, Michael Sprunk, Frank Schuster, Florian Ries, and Martin Haueis. On ambiguities in feature-based vehicle localization and their a priori detection in maps. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1192–1198. IEEE, 2019.
- [21] M. Sefati, Magnus Daum, Bjoern Sondermann, Kai Kreisköther, and Achim Kampker. Improving vehicle localization using semantic and pole-like landmarks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 13–19. IEEE, 2017.
- [22] Michael Himmelsbach, Felix Hundelshausen, and H. Wünsche. Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium (IV)*, pages 560–565. IEEE, 2010.
- [23] Julius Kümmerle, Marc Sons, Fabian Poggenhans, Tilman Kühner, Martin Lauer, and Christoph Stiller. Accurate and efficient self-localization on roads using basic geometric primitives. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5965–5971. IEEE, 2019.
- [24] Marc Sons, Martin Lauer, Christoph G Keller, and Christoph Stiller. Mapping and localization using surround view. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1158–1163. IEEE, 2017.

- [25] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.
- [26] Isabell Hofstetter, Michael Sprunk, Florian Ries, and Martin Haueis. Reliable data association for feature-based vehicle localization using geometric hashing methods. In *2020 International Conference on Robotics and Automation (ICRA)*, pages 1322–1328. IEEE, 2020.
- [27] Marc Sons and Christoph Stiller. Efficient multi-drive map optimization towards life-long localization using surround view. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2671–2677. IEEE, 2018.
- [28] R. Pepy, A. Lambert, and H. Mounier. Path planning using a dynamic vehicle model. In *2006 2nd International Conference on Information and Communication Technologies*, volume 1, pages 781–786, 2006.

Appendices

Appendix A

The hash table structure created during the training phase as mentioned in Section 2.2 is typically implemented using a dynamically-sized container with the best possible random-access time. Each entry in the hash table generally has a format similar to 'HashEntry' below. An example of what this may look like is presented in Figure A.1.

```
struct HashEntry
{
    // Hash value for this entry
    int hash;

    // Layer identifiers for each layer that contains a key at this hash value
    list<int> layers;
}
```

Along side the hash table, there must also be a database where the information about each basis is stored for later use. In Section 2.2, some of the information that might be found there was briefly mentioned. Below, the 'Basis' data structure provides more detail to what a possible implementation might look like. The database that each 'Basis' structure is contained in ultimately depends on the size and access needs of the implementation. In simple cases, a dynamically-sized array will suffice. An example of what this may look like is presented in Figure A.2.

```
struct Basis
{
    // Layer identifier (basically a serial number)
    int layer;

    // Location of the basis origin in map frame
    Point2D origin;

    // Distance between basis points (points that define the basis for this
    // layer)
    double dist;

    // Quantized distance between the basis points
    double dist_q;

    // Angle difference between the basis x-axis and map x-axis
    double rot;

    // Invariants in the basis frame
    list<Point2D> points;
}
```

```

// Variance on the invariants position in the basis domain
list<Point2D> variance;

// Quantized invariant positions
list<Point2D> keys;

// Feature IDs (index to corresponding mapped element)
list<int> feature_identifiers;

// Serial numbers for ambiguous constellations that contain this layer
list<int> ambiguity_identifiers;
}

```

Hash <i>Hash Value</i>	Bucket <i>Layer(s)</i>
1157	3, ...
1142	3, ...
1106	3, ...
1241	6, ...
1489	6, ...
1372	6, ...
...	...

Figure A.1: Example structure of Hash Table

Layer	Basis Pair	Model Points	Additional Info (Invariants, Keys, Variance)
1	1&2	1, 2, 3, 4, 5	{...}, {...}, ...
2	1&3	1, 2, 3, 4, 5	{...}, {...}, ...
3	1&4	1, 2, 3, 4, 5	{...}, {...}, ...
4	1&5	1, 2, 3, 4, 5	{...}, {...}, ...
5	2&3	1, 2, 3, 4, 5	{...}, {...}, ...
6	2&4	1, 2, 3, 4, 5	{...}, {...}, ...
...

Figure A.2: Example structure of Layers Database

In Section 3.2.1, groupings of quantized points that match between two or more bases were introduced as similarities. By exhaustively searching through all combinations of matching keys across all combinations of layers, a complete database of similarities is obtained. The database is sorted by the number of points belonging to the similarity (*i.e.* the vertex count, K). At each vertex count, a dynamically-sized amount of similarities are stored as shown in the text block below.

```

K | list(Similarities)
-----
4 | sim1, sim2, sim3
3 | sim4, sim5
2 | sim6, sim7, sim8
1 | sim9, sim10, sim11, sim12

```

Each similarity in the database is a data structure that could be implemented as shown below in 'Similarity'.

```

struct Similarity
{
    int vertex_count;           // Number of points in this similarity
    int layers[2];             // Layers (bases) that created this
                               // similarity

    list<Point2D> lhs_points;    // Invariants from the left-hand layer
    list<Point2D> rhs_points;    // Invariants from the right-hand layer

    // Discretized invariants
    // (Recall, these are the same between the two layers, this is why it's a
    // similarity)
    list<Point2D> keys;

```

```

    // Feature IDs for the matched (similar) points
    list<Point2D> similar_features;
}

```

After obtaining the complete list of similarity sets organized by vertex count, Section 3.2.2 suggests that the non-unique information from the similarity sets can be condensed into constellations of points tying back to a unique set of layers. This step removes a great deal of redundant information as well as enables a better format for computing other useful information that can be used during the recognition step. Each ambiguous constellation has a data structure that looks similar those shown below in 'AmbiguousConstellation' and the sub-structures 'AmbiguousLayer' and 'AmbiguousTransform'.

```

struct AmbiguousConstellation
{
    int serial_number;           // Unique identifier for this constellation
    int vertex_count;           // Number of points in this constellation

    // Number of occurrences of this constellation in map (same as the number
    // of layers)
    int occurrences;

    // Discretized invariants that are shared between all layers
    list<Point2D> constellation;

    // List of layers and their respective info that belong to this ambiguity
    list<AmbiguousLayer> layers;

    // List of transformations between the constellation map locations
    list<AmbiguousTransform> tfs;
}

```

```

struct AmbiguousLayer
{
    int layer;                   // Layer identifier

    // Invariants that belong to this layer AND the ambiguity
    list<Point2D> points;

    // Feature IDs that correspond to the points above
    list<int> feature_identifiers;

    // Centroid of the cluster of ambiguous points in the map frame
    Point2D centroid;

    // Angle difference between the constellation x-axis and map x-axis
    double orientation;
}

```

```

struct AmbiguousTransform

```

```

{
// Layer identifiers that make up this transform
int layers[2];

// Translational offset between the two centroids of the constellation pairs
double translation;

// Rotational offset between the two constellation x-axes
double rotation;
}

```

Lastly, an rough example of what an AmbiguousConstellation structure may look like when populated is shown in the text block below.

```

Am1:
-> serial_number = 1;
-> vertex_count = 4;
-> occurrences = 2;
-> constellation = { [0,2], [0,-2], [4,7], [-3,-5] };
-> layers = {
  lyr1:
    layer = 7;
    points = { [0.4, 1.8], [0.3, -2.2], [4.1, 6.9], [-2.8, -5.0] };
    feature_identifiers = { 2, 7, 19, 32 };
    centroid = [Cx7, Cy7];
    orientation = R7;
  lyr2:
    layer = 11;
    points = { [-0.1, 1.7], [-0.3, -1.9], [3.8, 6.8], [-3.2, -4.8] };
    feature_identifiers = { 4, 9, 13, 21 };
    centroid = [Cx11, Cy11];
    orientation = R11;
}
-> tfs = {
  tf1:
    layers = { 7, 11 };
    translation = [Cx11-Cx7, Cy11-Cy7];
    rotation = wrapPi(R11-R7);
}

```
