2018

Positioning by Road Feature Correspondence

Tyson Demarest
University of Rhode Island, tyson.demarest@gmail.com

POSITIONING BY ROAD FEATURE CORRESPONDENCE

BY

TYSON A. DEMAREST

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2018

MASTER OF SCIENCE THESIS

OF

TYSON A. DEMAREST

APPROVED:

Thesis Committee:

Major Professor	Jean-Yves Hervé
	Lutz Hamel
	Peter Swaszek
	Nasser H. Zawia DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2018

ABSTRACT

This research analyzes a new method to localize a moving vehicle within a known road network using differential odometry and a digital road map. The technique proposes geometrically hashing road features whose curvature is great enough to represent a distinct, measurable, real-world phenomenon. Several other strategies using road map data to constrain a localization search are discussed. In the recognition phase, this research proposes using a modified particle filter provides a way to maintain, test, and resample multiple hypotheses of the vehicle's dead-reckoned location.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Jean-Yves Hervé for his extremely valuable guidance, his generous patience, and his insightful lectures. He is an institution in his own right, and his friendly presence and keen instruction are greatly appreciated by the entire Department of Computer Science and Statistics.

I would also like to thank Dr. Lutz Hamel for his excellent lectures, many of which inspired this research, and for his generosity in providing me with the opportunity to address the programming language considerations specific to my approach. Thanks also to Dr. Peter Swaszek for agreeing to be a member of my committee and to Dr. Gérard Baudet for his inspiring lectures and scholastic guidance. Additionally, I would like to thank Department Secretary Lorraine Berube for her friendly help in preparing and filing the necessary forms with the University. I would also like to thank Dr. Marc Hutchison for his friendly, encouraging advice and for being the chairman of my defense committee.

Finally, I would like to thank my lovely wife Elyse for her inspiration and patience. Localization is not a difficult problem when she is near.

Contents

List of Figures

List of Tables

LIST OF ACRONYMS AND ABBREVIATIONS

ABS Anti-lock Braking System

CUKF Constrained Unscented Kalman Filter

CAN Controller Area Network

DGPS Differential Global Positioning System

EKF Extended Kalman Filter

GPS Global Positioning System

GNSS Global Navigation Satellite System

IMU Inertial Measurement Unit

INS Inertial Navigation System

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute

MCL Monte Carlo Localization

OBD On Board Diagnostic

OSM OpenStreetMaps

PID Parameter ID

SLAM Simultaneous Localization and Mapping

UKF Unscented Kalman Filter

CHAPTER 1

Introduction

1.1 Problem Context

Current positioning techniques fall within one of two categories: 1. Dead-reckoning (sometimes written as ded-reckoning) where position is deduced metrically from previously known positions, and 2. Position Fixing, where position is determined by relation to infrastructure or landmarks [1]. Below is a brief summary of each.

With dead-reckoning a navigator determines his or her position by accumulating movements, i.e. displacements in position, and adding them to the last known position. An everyday instance of dead-reckoning is the use of step-by-step directions to a lost motorist.[1] Here is a colloquial example: "Go to the stop sign and take a right. Drive three blocks and the post office is on your left." The starting position is where the conversation took place, and the change in position is given. A distance measure, which in this example is the number of blocks, creates a spatial reference frame in which odometry plays a necessary role. To use the directions the navigator must count blocks. Additionally, because the navigator in the example is on a surface and not a merely a line, the navigator must also keep track of heading.

The second type of navigation is position fixing. Position fixing determines one's absolute location by using environmental or infrastructural indicators. This navigation technique does not start with a known position but aims to resolve it from surrounding fixtures. In the above example, the lost motorist might be aware of the relative position of a shopping center and stop to find its absolute position

[1]As an aside, turn by turn instructions are known as "progressive taxi instructions" in the aviation community [2].

on a map. The shopping center is the landmark and the navigator's recognition of it instigates a qualitative event [3]. Additionally, humans associate these kinds of landmarks with their adjacency to other landmarks in a topological way. In the above example, it is unlikely that the direction-giver links each landmark in those directions by some geometric algorithm, but rather through a topological connectedness, the memory of which is enhanced with numbers for precision. The direction-giver might have thought before speaking, "If I get to the stop sign, then I'll be close to the post office. If I take a left there I'll get to the post office in what feels like about three blocks."

A more sophisticated and popular example of position fixing is Global Navigation Satellite Systems (GNSS), such as GPS, in which satellites transmit details about their own position and time signals to navigators on the ground. In GNSS the ground receiver uses the distance from each GNSS satellite to trilaterate its own position. GNSS is an example of infrastructure-based positioning, which requires a pre-existing man-made system in place.

1.1.1 Distinctive Places and Features

Kuipers and Byun in [4] define a "distinctive place" as "the local maximum found by a hill climbing control strategy, given an appropriate distinctiveness measure." However, we will relax this definition to be "the location of an appropriately high distinctiveness measure." We do this for two reasons. First because the research here describes a passive system, where no control inputs influence the system and position is estimated only after odometric measurements are made, we do not have the ability to "hill climb" to maxima. Second, as will be described in the Methodology Section, because we quantify the curvature of radius of the road to which a vehicle is bound, we cannot maximize this quality even by driving it in a slightly different way.[2]

To convey the importance of using landmarks and "distinctive places" in any global localization strategy, it is instructive to examine techniques that do not use features with obvious distinctiveness, but through automation recognize distinctive features even still. Such an example is Terrain Reference Navigation (TRN), in which environmental features sampled (typically by radar) belong to set of elevations that exist on a continuous domain, i.e. the set real numbers between the lowest and highest terrain [1]. The navigator using TRN continues to accumulate elevation measurements and their relative positions until a sufficient terrain signature is generated and matched with a previously acquired database of terrain elevations. The words "sufficient terrain signature" should evoke the same idea of landmarks. The time series elevation data, despite being on a continuous domain and uninteresting to humans, becomes enough to distinguish itself from other possibilities in a known dataset.

1.1.2 Topometric Positioning

The positioning solution proposed in this research derives from a human's ability to navigate with turn-by-turn directions and simultaneously to locate oneself by recognizing landmarks along the way. This instinctual approach to human positioning forms a subtle nexus between geometric navigation environments where spatial relations determine one's position in a reference frame, and qualitative or topological navigation environments which highlight discrete landmarks whose adjacencies form a navigable graph. The fusion of these environments has been called topometric localization [5].

[2]We remark here that although measuring the path of a vehicle during a lane change in the middle of a turn would indicate a *different* curvature than if the vehicle did not change lanes during the turn, we note that this does not maximize the distinctiveness of the road's curve, and instead effectively adds noise to the measurement of it.

1.2 Significance of the Study

For most people satellite navigation is the sole solution to the global positioning problem. However, alternative positioning strategies can augment GNSS in several common situations, and provide a safeguard against GNSS outages. The following subsections describe situations where a solution to road-based GNSS-free localization might be useful.

1.2.1 Commercial Road Navigation

Global Positioning System authorities including The United States Air Force guarantee ongoing availability and maintenance of American GPS service [6]. Nevertheless, GNSS signals are easily jammed, interfered with, or blocked by weather because they are relatively weak [7]. Accounting only for free-space dissipation, the received signal's strength is approximately -120dBm at the earth's surface [7]. Additionally, a GPS receiver demands even greater signal strength than that required for tracking, and a GPS receiver may demand a even more time to get an initial position in bad weather.

On the other hand, intrinsic sensors such as vehicle odometers can immediately begin providing data to use for estimating location, despite any natural or man-made radio frequency interference.

1.2.2 Defense Road Navigation

Although mention of military positioning systems conjures up the image of soldiers, naval vessels, and fighter jets, there are many use-cases of military positioning that are similar to their civilian counterparts. Road-based vehicle patrols or re-supply routes both require positioning that is constrained to a road network, and both might take place in a GNSS-denied environment [8]. Further, the threat of GNSS failure impinges most greatly those who are least trained to navigate without it. Whereas the loss of GNSS would affect pilots, sailors, and expert soldiers to some degree, such a loss might cripple some other military units' ability to navigate foreign roads and arrive on time.

1.2.3 Extension to Indoor Localization

With modification this research can be applied to indoor positioning. Indoor positioning is an active area of study among the major technology companies, and a lot of resources are being spent to bring indoors the high level of positioning service that is offered outside by GPS [9].

1.2.4 Game Non-Player Character Positioning AI

In many games a non-player character (NPC) is imbued with an automatic awareness of location. On the other hand, the player is often forced to find himself within the game world through trial and error. The technique researched here could be cheaply integrated into the artificial intelligence of NPCs and give them a more realistic ability to learn their locations. Here, the "mapping" part of Simultaneous Localization and Mapping (SLAM) might or might not belong well to the game story because the player or NPCs could already have a complete map at their disposal [10, 11].

1.3 Related Work

Most of the current techniques in the field of map-based global localization can be categorized as either "map constrained tracking" or as "landmark recognition." Below is a brief overview of each, followed by a closer look another even more pertinent strategy.

1.3.1 Map Constrained Tracking

Most of these tracking techniques do not use roadmap data directly to address the "kidnapped robot" problem; rather, they rely on an outside source of global location information such as GPS or an otherwise previously known location. Only after which point does the vehicle use dead-reckoning to move geometrically about the state space and provide it with a continuous estimate of the position. When digital map data is consulted in combination with this technique, the drift from dead-reckoning is corrected to provide an even more accurate estimate. Usually some form of map-matching technique, wherein a position estimate is given to the algorithm as input, and a map edge or node is returned, is used.[3]

In [12] a survey of techniques are discussed, but the two relevant ones here are detailed in the subsections "Dead reckoning: Map matching" and "Dead reckoning: Dynamic filtering."[4] With these methods, a position is known at initialization and is a) updated by the position on a map corresponding with a dead-reckoned change in position or b) updated to be consistent with a motion model which is itself corrected by additional measurements. In their research Karlsson and Gustafsson only use the Anti-lock Braking System (ABS) Wheel Speed Sensor data from the vehicle's Controller Area Network (CAN) bus to perform differential odometry, although techniques using an IMU are discussed. As another important aside, the authors also use the term "virtual sensor" when referring to measurements provided by consulting the road map data, and we adopt that term in this research.

In [13] a similar technique using differential odometry, roadmap data, and a particle filter is discussed. The experiment is divided in the tracking task, i.e. initial position is known, and the global localization task. For tracking it is unclear from the extended abstract how the road map data is traversed by hypothesis particles, but it is likely the same map matching style used in [12] because of Merriaux's reference to similar work by Gustafsson. For localization, a much greater

[3]The sensors used for dead-reckoning are usually some combination of wheel speed sensors, an Inertial Measurement Unit (IMU), a camera for visual odometry, or active range finder

[4]Karlsson and Gustafsson later discuss a Particle Filtering approach which inspired the research here.

number of particles are spread uniformly across the search space, in this case a road network of approximately 100km. With 3600 particles spread across this map, the authors claim that global location is determined within 90 updates, each of which taking 0.51s on a single core. Additionally, the authors identify a linear relationship between runtime required to update all particles and the number of particles.

In [14] Li and Leung propose a Constrained Unscented Kalman Filter (CUKF) to fuse Global Positioning System (GPS), Inertial Navigation System (INS), and digital map data. While GPS data provides the drift-free global localization capability of their system, the INS provides dead-reckoned position estimates with higher resolution. In particular they use a measure of agreement of the vehicle's heading with the direction of roads in a neighborhood in order to further constrain position estimates. They show that the CUKF significantly improves the position estimates over a standard unconstrained Unscented Kalman Filter (UKF).

In [15], Najjar and Bonnifait directly address the issue of how best to select a road to solve the road matching problem.[5] Their approach consists of a Multi-sensor Fusion Extended Kalman Filter (EKF) with Differential Global Positioning System (DGPS) and ABS wheel speed odometry as inputs. This position model then feeds a set of road selection process whose criteria are based on 1) proximity to the estimated position and 2) consistency with travel direction (as in [14] with speed as an additional consideration for this consistency. In particular, the possibility of a vehicle traveling at 40kph being on a road perpendicular to its trajectory is highly unlikely and Najjar and Bonnifait's model takes these sorts of constraints into account.

[5]The road matching arises when global positioning data such as that from Global Navigation Satellite System (GNSS) is too noisy to stay on a road and must be corrected to represent a position on digital map data, especially GPS turn-by-turn navigation devices.

Summary

Landmarks *distinguish* roads so that each road is measured as in a way that separates it spatially from other roads. Because of the limited degree of landmark characterization in the above techniques, we classify these only as "map constraining," and not as "feature detection" or "landmarking." In the next subsection we discuss several techniques that make more overt use of landmarking in order to determine global location.

1.3.2 Landmark Recognition

The most efficient techniques for global localization apply Kuipers's idea of a "distinctive place" [16] by fixing landmarks to an either learned or already known map. If a known location is recognized to be close, the degree of confidence in a navigator's position can be restored to the degree of confidence held in the position of the landmark. In some applications such as [11] the landmarks form a graph whose measure of connectedness is the ability of a game character to navigate from one landmark to another. In [11] Schraffenberger and Hervé call this topology a "Perceptual Navigability Graph." The approaches discussed below apply the use of landmarks to spatially distinguish roads (or in some cases spaces) with varying measurable features.

In [17], Jensfelt and Kristensen present a technique that consider multiple hypotheses by generating a set of possible poses when a known visual feature is recognized. For example when their robot recognizes a door, a set of hypotheses consistent with seeing the door under the circumstances, e.g. location of doors previously known to be on the wall, is generated. Each hypothesis, similar to particles in the particle filter, is updated with its own Kalman Filter estimates and covariance matrices. Emphasis is placed on topology and map discovery. Notably in this research odometric dead-reckoning and feature recognition are reconciled by a process the authors term “data association,” wherein the authors threshold a similarity measure given by squared Mahalanobis distance between the pose hypothesis and the pose “candidates.” We assume the convention of naming feature-generated locations as “candidates” in this research, as well as assuming the terms coined by the authors in [17] to describe “supportive” estimates as generated by dead-reckoning and “creative” estimates to be those generated by feature recognition.[6]

In [5], Badino *et al.* also use images as recognition features along a navigable path. In this case the research is performed outside with images similar to those in Google Streetview, and range measurements are taken to accompany the images, all taken periodically along a fixed 8km route. This route comprises a discrete state space and topology of landmarks. In the query phase the authors use their novel U-SURF and WI-SURF features to update a Bayesian filter. This technique was 100% effective in its ability converging to the correct global location. Another notable detail is that the researchers in [5] created their image training set over all four seasons in order to make querying resilient against changes in vegetation, lighting, etc.

In [18] Floros *et al.* estimate the shape of a road by visual odometry and match it with OpenStreetMap data. In their project they call “OpenStreetSlam” the authors use Harris corners and RANSAC to update a Kalman Filter which updates a camera’s pose estimate and, ultimately, the shape of a driven path. The researchers make use of Monte Carlo Localization (MCL), or particle filter, in order to maintain a distribution of possible locations. They update their hypotheses using measurements of their query shape against the road data with Fast Directional Chamfer Matching across the entire set of possible roads. This is done

[6]Although in [17] the authors were also mapping the environment, we retain the notion of a “creative” update because it is still one that could potentially create new hypotheses, and otherwise provides additional confidence in hypotheses that already exist.

online while the vehicle is traveling a required initial distance and takes approximately 11.5s over the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) vision dataset consisting of a total driving distance of 39.2km within a 1km radius [19]. The basic components of this approach, i.e. shape of a traveled road as determined by odometry, OpenStreetMap data, particle filter with updates based on shape correspondence, amount to a vast similarity in the research that will be described here. We remark here that the work in [18] differs substantially from ours in that Floros *et al.* use visual odometry to detect roadmap features and use line segments to represent them.

1.3.3 Geometric Hashing and Navigation

In their paper "Image-Based Navigation on a Chip," Lifshits *et al.* propose a navigation technique to locate the end effector of robotic systems used for electronic chip manufacturing functions such as "lithography, cutting and inspection" [20]. Given a known wafer map, the authors propose geometrically hashing all of the features of the wafer map in a preprocessing stage, and subsequently locating the robotic eye-point by geometrically hashing a subset of the features in a lookup phase[7]. Although "navigation" here does not refer to that of a surface-bound vehicle, the essential components of locating oneself with a limited perspective of a known overall environment, a theoretical approach discussed in detail by Kuipers in [21], still apply. Further, the research in [20] is the only previously existing work to use geometric hashing to solve the "kidnapped robot" problem. As a difference to Lifshits *et al.*'s work and the work here, we want to emphasize that our approach considers graph-like data, and not a point cloud. This distinction significantly changes the nature of what comprises a feature and how those features are discovered.

[7]Section 1.3.4 provides an overview of the classical geometric hashing technique used in [20].

1.3.4 Geometric Hashing Overview

The technique of geometric hashing has been used in many fields of research to align a segment of graph data with its original set. Applications can be seen in protein structure alignment [22] digital photograph registration [23], and astrometry [24].

The technique of geometric hashing can be broken down into two phases: building a hash table (the Training Phase) and indexing test points within the hash table to determine to which basis the points belong (the Recognition Phase)[25].

When building the geometric hash table, the goal is to create key-value pairs whose keys represent test points in the coordinate system produced by the corresponding value, the coordinates in whose basis the keys exist. For example, "1.05,-0.35": (35143,85243: 35100,84987) represents a point whose coordinates are 1.05, -0.35 after it has undergone the change-of-basis transformation defined by the two points 35143, 85243 and 35100, 84987. These two points are recorded in the original coordinate system so that when the lookup phase takes place, the result will be in a coordinate system useful to the user.

These key-value pairs are constructed by examining combinations of two points on the model, creating a change-of-basis function using the two points as a new basis, and then recording the coordinates of the remaining points in the new coordinate system into the hash table. Below is a more thorough description.

An important *caveat* to note is that a query pattern may be matched to more than one location in a set. Take for instance the short path consisting of a straight drive forward for a trivial distance. This could be matched to any small straight road portion of the map, whose matching paths will be many. Rather, the technique of matching the geometry of a path to a map requires that the path contain enough information, here unique points, to separate it from similar possible paths.

The rest of the first phase is straightforward: a transformation C is constructed for each combination of two points in the model, and with that change-of-basis operator, new coordinate values are calculated for the remaining points in the model. The coordinate values are then inserted into the hash table as the keys, and the basis as the value. If there is a collision, the additional bases are appended and the value item forms a list. The matching phase is similar to the building phase. Once a traveled set of test points is produced, a change-of-basis is created with two of the appurtenant points. The rest of the traveled points are transformed into the new basis, and their values are looked up in the hash table generated from the model in the first step. All basis matches are recorded and appended to a list. Then a voting scheme where the basis pair that is most represented on the list determines the winner from the list. In application, these two steps require significant attention to peculiarities among the data and computing limitations. In the next section we will discuss in details specific to Path Registration how our implementation deviated from the overall geometric hashing strategy. [26]

1.4 Thesis Statement

This research analyzes a new method to localize a moving vehicle within a known road network using differential odometry and a digital road map. The technique proposes geometrically hashing road features whose curvature is great enough to represent a distinct measurable real-world phenomenon. In order to withstand uncertainty and absence of measurement events, a modified particle filter provides a way to maintain, test, and resample multiple hypotheses of the vehicle's dead-reckoned location.

List of References

[1] P. D. Groves, *Principles of GNSS, Inertial, and Multi-Sensor Integrated Navigation Systems Second Edition*. Boston: Artech House, 2013.

[2] "Progressive Taxi Instructions," 2017. [Online]. Available: https://www.skybrary.aero/index.php/Progressive_Taxi_Instructions

[3] P. Ranganathan, J.-B. Hayet, M. Devy, S. Hutchinson, and F. Lerasle, "Topological Navigation and Qualitative Localization for Indoor Environment Using Multi-Sensory Perception," *Robotics and Autonomous Systems*, vol. 41, no. 2-3, pp. 137–144, 2002.

[4] B. Kuipers and Y.-T. Byun, "A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations," *Robotics and Autonomous Systems*, vol. 8, no. 1, pp. 47 – 63, 1991.

[5] H. Badino, D. Huber, and T. Kanade, "Real-Time Topometric Localization," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA, May 2012, pp. 1635–1642.

[6] National Coordination Office for Space-Based Positioning Navigation and Timing, "Gps.gov: Frequently Asked Questions," 2016, Accessed: 2016-2-21. [Online]. Available: http://www.gps.gov/support/faq/#off

[7] NXP Semiconductors, "GPS, LNA, Sensitivity, Jamming, Cohabitation, TTFF," Eindhoven, 2009, Accessed: 2016-2-21. [Online]. Available: http://www.nxp.com/documents/other/75016740.pdf

[8] C. Hoffman, "China's Space Threat: How Missiles Could Target U.S. Satellites," 2009, Accessed: 2016-2-21. [Online]. Available: http://www.popularmechanics.com/space/satellites/a1782/4218443/

[9] A. Schutzberg, "Ten Things You Need to Know about Indoor Positioning," 2013, Accessed: 2016-2-21. [Online]. Available: http://www.directionsmag.com/entry/10-things-you-need-to-know-about-indoor-positioning/324602

[10] N. K. Dhiman, D. Deodhare, and D. Khemani, "A review of path planning and mapping technologies for autonomous mobile robot systems," in *Proceedings of the 5th ACM COMPUTE Conference: Intelligent Scalable System Technologies*, ser. COMPUTE '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:8. [Online]. Available: http://doi.acm.org/10.1145/2459118.2459121

[11] M. Schraffenberger and J. Y. Hervé, "Agent Abilities in a Landmark-Based Mapping Model," in *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, Oct. 2006, pp. 2493–2498, Taipei, Taiwan.

[12] R. Karlsson and F. Gustafsson, “The Future of Automotive Localization Algorithms: Available, Reliable, and Scalable Localization: Anywhere and Anytime,” *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 60–69, March 2017.

[13] P. Merriaux, Y. Dupuis, P. Vasseur, and X. Savatier, “Wheel Odometry-Based Car Localization and Tracking on Vectorial Map,” in *Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, Oct. 2014, pp. 1890–1891.

[14] W. Li and H. Leung, “Constrained Unscented Kalman Filter Based Fusion of GPS/INS/Digital Map for Vehicle Localization,” in *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, vol. 2, Shanghai, China, Oct. 2003, pp. 1362–1367.

[15] M. E. B. E. Najjar and P. Bonnifait, “Road Selection Using Multicriteria Fusion for the Road-Matching Problem,” *Proceedings of the IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 279–291, June 2007.

[16] B. J. Kuipers and Y. Byun, “A Qualitative Approach to Robot Exploration and Map-Learning,” in *Workshop on Spatial Reasoning and Multi-Sensor Fusion*, St. Charles, IL, USA, Oct. 1987, pp. 390–404.

[17] P. Jensfelt and S. Kristensen, “Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, Oct. 2001.

[18] G. Floros, B. van der Zander, and B. Leibe, “OpenStreetSLAM: Global Vehicle Localization Using OpenStreetMaps,” in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2013, pp. 1054–1059.

[19] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision Meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[20] M. Lifshits, E. Rivlin, and M. Rudzsky, “Image-Based Navigation on a Chip,” in *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference*, vol. 1, Como, Italy, May 2004, pp. 504–509 Vol.1.

[21] B. J. Kuipers, “Representing Knowledge of Large-Scale Space,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977.

[22] N. Leibowitz, Z. Y. Fligelman, R. Nussinov, and H. J. Wolfson, “Multiple Structural Alignment and Core Detection by Geometric Hashing,” in *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, Heidelberg, Germany, 1999, pp. 169–177.

[23] U. Bhosle, S. Chaudhuri, and S. Dutta Roy, "A Fast Method for Image Mosaicing Using Geometric Hashing," *IETE Journal of Research*, vol. 48, no. 3-4, pp. 317–324, 2002.

[24] C. Harvey, "New Algorithms for Automated Astrometry," Ph.D. dissertation, University of Toronto, Toronto, Ont., Canada, 2004.

[25] H. Wolfson and I. Rigoutsos, "Geometric Hashing: An Overview," *IEEE Computational Science and Engineering*, vol. 4, no. 4, pp. 10–21, 1997.

[26] A. G. Konheim, *Hashing and the Secure Distribution of Digital Media.* Hoboken, NJ, USA: John Wiley and Sons, Inc., 2010, pp. 320–323. [Online]. Available: http://dx.doi.org/10.1002/9780470630617.ch19

CHAPTER 2

Methodology

Although only one of the modules (RoadMapHash, which preprocesses the OpenStreetMaps data into a key-value database of recognizable curvature features) has been fully implemented, we propose a control system architecture that enables a vehicle to determine its location both on-line for use in consumer facing applications, and off-line, in applications where the vehicle data is merely recorded for later analysis by a management authority.

2.1 Architecture of the Positioning Strategy

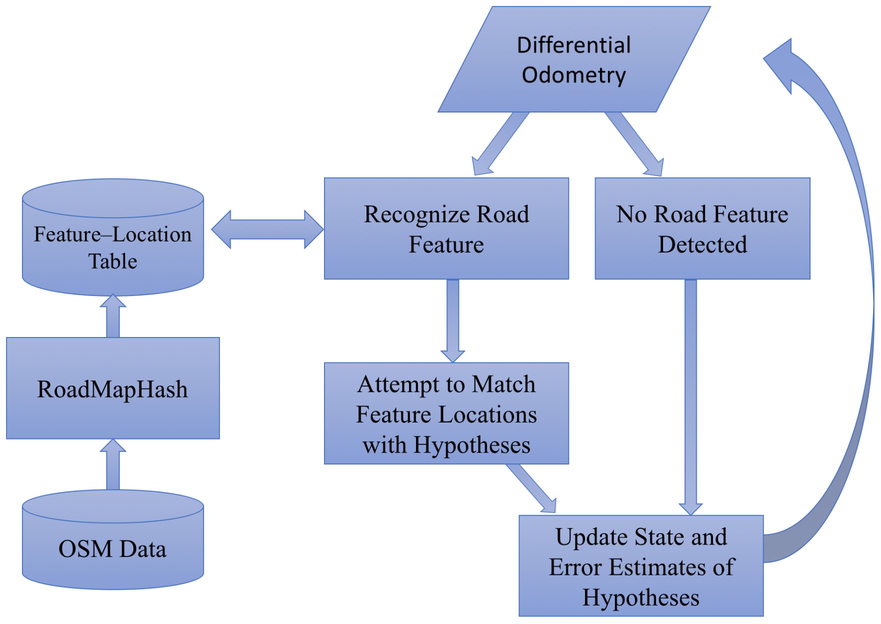


Figure 1. Overall System Diagram

The components of the RoadMapLocate system are as follows. First, RoadMapHash preprocesses the OpenStreetMaps data into a key-value database of recognizable curvature features. Second, the RoadMapTrack sub-system, now running on an Arduino Uno R3, reads the rear Wheel Speed Sensor data from the vehicle's CAN bus. This data may rest in a data store, such as an SD card, for future off-line processing, or it may be fed directly to the RoadMapFilter system, which performs the Monte Carlo Localization algorithm described in Section 2.7. This last component analyzes the vehicle update data to identify high curvature features, and identifies their signature in the RoadMapHash database. This database may be used on-line as a file store or transfered over a network connection. Ultimately, the RoadMapFilter algorithm returns a best hypothesis to the user.

In the next sections we will set up the vocabulary necessary to discuss the particularities of RoadMapHash, cover the sensors and data we use in this research, propose a new data-centric language for conveying road curvature data, analyze RoadMapHash, and finally provide an overview of the future components of the overall RoadMapLocate system.

2.2 Definitions

Definition 2.2.1. Node

A node is a point on the Earth's surface lying on a road. Nodes are the vertices in the road network topology. They are described both in geographic coordinates on the 1984 World Geodetic System (WGS84) ellipsoid and in Universal Transverse Mercator coordinates on a projected flat-earth model.

Definition 2.2.2. Way

A way is a list of nodes connected by a whole or partial road, which is a real-world drivable surface. The Nodes appear in order as one would drive them. Ways are assumed to be drivable in two directions unless the road is specifically marked as oneway in the data [1]. (Additionally, by specification of OSM, Ways cannot overlap. This constrains the graph of nodes as a planar graph, the consequence of which we profit from in Subsection 2.1.22.)

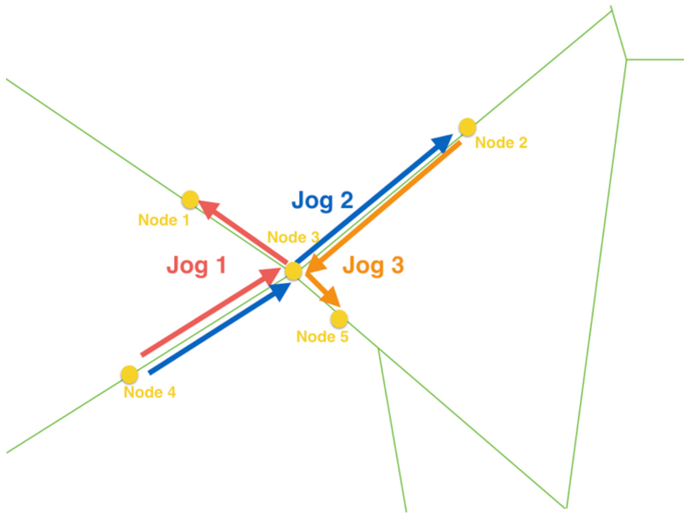


Figure 2. Example Jogs and their Nodes

Definition 2.2.3. Jog

A Jog is an ordered triple of nodes that represents three points connected by a road path. It can be traveled from the first node through the second node to the third node.

Remark. A Jog is not bidirectional. If three Nodes n_1, n_2, and n_3 belong to ways indicating travel is permitted both from n_1 to n_2 to n_3 as well as from n_3 to n_2 to n_1, then two unique Jogs must represent each navigable path.

A Jog may not begin and end at the same node. In other words, U-Turns are not considered as drivable paths.

The Jog structure is useful when calculating the change in direction, road curvature, and distance along a path of connected nodes.

Remark. See that the set of Jogs J_C traversing through a center node n_C is a subset of the Cartesian product of the set N_{in} of nodes, for which there exists a way in-bound to n_C, with the set N_{out} of nodes, for which there exists a way out-bound from n_C. Because U-turns are not included in the definition of a Jog, J_C is usually a proper subset of this Cartesian product, and does not include the set of pairs where the start node n_S is the same as the end node n_E.

Definition 2.2.4. Sharp Jog

A Jog is "sharp" with respect to a given radius of curvature if the circle circumscribing the Jog has a radius less than the given one. On a map these look like what we think of when we say "Turns" but because of the challenges in sensing the phenomenon associated with Turns, a sharp Jog has only this constraint.

Definition 2.2.5. Turn

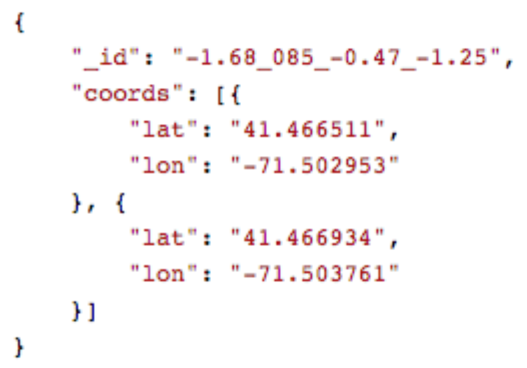
A Jog with sufficient curvature might be considered a Turn in the simple case above; however, in general a Turn must meet additional criteria beyond being such a Jog for several practical reasons. The most important of them is that a road network may include chains of high-curvature Jogs, each of which may not be in isolation considered a unique Turn. For instance, a mile-long elliptical race track may be denoted on a map by hundreds of Jogs, but a Turn event might only be measurable twice per lap, once at each curve. Instead of merely checking Jogs for high curvature, there must be some thresholding put in place to determine where a Turn starts and stops, and a Turn must include the possibility of incorporating more than one Jog.

Definition 2.2.6. TurnEvent

TurnEvents are for the basis of the geometric hash map key and are combinations of two Turns.

Definition 2.2.7. Jog Curvature

The radius of curvature R_J of a Jog consisting of three Nodes N_S, N_C, and N_E is equal to the radius of the circumcircle of the triangle $\triangle N_S, N_C, N_E$.



```
{
    "_id": "-1.68_085_-0.47_-1.25",
    "coords": [{
        "lat": "41.466511",
        "lon": "-71.502953"
    }, {
        "lat": "41.466934",
        "lon": "-71.503761"
    }]
}
```

Figure 3. Example Turn Event Key-Value Pair

Remark. By the definition of radius of curvature,

Let $\overline{B_{S,C}}$ denote the perpendicular bisector of the line segment from N_S to N_C, and let $\overline{B_{C,E}}$ denote the perpendicular bisector of the line segment from N_C to N_E.

We remark that the intersection of $\overline{B_{S,C}}$ and $\overline{B_{C,E}}$ is equidistant to each of N_S, N_C, and N_E and thus the center of the circumcircle of $\triangle N_S, N_C, N_E$.

2.3 Sensors and Data

2.3.1 Odometric Sensor

In this research we intend to use the ABS wheel speed sensors to record the differential odometry of the moving vehicle[1].

We refer here to [3], which makes the following remark on the accuracy of ABS wheel speed sensors.

> Typical accuracy of ABS wheel speed sensors over OBDII is 0.025 meters/second, where as the accuracy of the transmission rotation sensor is approximately 0.28 meters/second.

Although ABS wheel speed sensors are accurate and can be used to perform differential odometry, they are unfortunately difficult to interface with. Whereas

[1]For a benchmark dataset we will also test our technique on the odometry data provided by [2]. This data provides parameters given to a ClearPath Husky robot and must be modified slightly to fit into the RoadMapTrack system.

vehicle speed (as measured from the transmission) is part of the "Mode 1" On Board Diagnostic (OBD) interface, the OBD request codes for other systems such as ABS are not openly published, and often come at a significant cost for scan tool manufacturers.

Researchers use several strategies to determine these proprietary Parameter ID (PID) codes[2]. An step-by-step overview of one of our strategies is given below.

1. Build a DIY scan tool with an Arduino Uno R3 and a CAN Bus Shield.

2. Brute force-scan the CAN bus for responses and record the PIDs that provided a response.[3]

3. Write an Arduino sketch that queries the responsive PIDs and records their response.

4. Drive the vehicle in a known pattern while recording the values. A useful pattern is a circle as in a roundabout. This will provide two different rear wheel speeds that are generally constant during the evaulation period.

Figure 4 shows the response from the PID code 0x2103 [4] indicating that a sensor reading was zero, increased in value during a reverse movement, returned to zero, increased during a forward movement, and finally settled at zero.

2.3.2 Map Data

This research uses the OpenStreetMap dataset. The OpenStreetMap (OSM) dataset is an open source competitor to Google Maps and provides direct access to geospatial vector data via its Planet.osm data file. The OpenStreetMap Markup Language, denoted by the ".osm" file extension, is a spatial implementation of

[2]Wikipedia offers a very informative overview of the OBD2 protocol at [4].

[3]Here the search can be limited by knowing the manufacturer's specific PID header. Toyota's is 21.

[4]A response for PID 21 would be 0x40 greater than the code itself, so 0x61.

XML and provides a convenient way to define geographic types, especially roadways, and assign attributes to them. As in other vector file formats, nodes are given in a coordinate system, and edges are implied by the inclusion of multiple nodes within a shape. In OSM, the shapes are linestrings and polygons. Road Selection: [5] succinctly summarizes the issues when selecting roads for localization:

> This problem ... of selecting the "good" segments from the subset $\{S1, ..., Sn\}$ ... is difficult because of several factors.
>
> - The position is estimated with errors that can be magnified by multipath effects. In addition, the transformation between the GPS coordinates (World Geodetic System 1984 reference system) and the projection frame of the map [here, the French Nouvelle Triangulation Française (NTF) Lambert coordinate system] can induce errors.
> - The coordinates of the segments are falsified by errors due to terrain measurements that are carried out by cartographers as well as due to numerical approximation.
> - The road network in the database does not always correspond to reality, i.e., it can contain old roads that no longer exist or new roads that are not yet in the database.
> - The map does not contain all road network details. For example, a roundabout can be represented as a simple point.
> - The vehicle is moving on a 3-D surface, whereas the map represents a planar view.
> - The vehicle does not travel exactly on the segments representing the roads.

2.4 Data Centric Programming Language

2.4.1 Introduction

Data Centric Programming Languages form a nexus between data and computation. The focus of a data language's use coheres with its structure, and with the rise of network analysis, graph-centric data languages have recently achieved first class status [6, 7, 8]. In this research we devise an extended graph data language in order to achieve three main goals. Primarily, the new language will provide a means to serialize special data structures used in the research. Second, and more importantly, the new data specification, in binary form, will form a basis for a self-referencing database engine. Finally, distributed execution environments, such as MapReduce, will benefit from the JogML and JogBin formats.

2.4.2 Jog Modeling Language (JogML)

JogML Introduction

Existing graph data languages have three features in common. One, they emphasize universality with commonly used data specifications, such as XML. Two, they provide extensibility for arbitrary types and attributes, such as with tags or attributes. These features are unnecessary for this research because this research provides its own data manipulation tools. And three, most current graph languages are limited in their ability to describe hyper-edges [9]. This is unsuitable for this research because it uses unique data structure called a "jog," which is a sub-graph consisting of three nodes. Thus, the data language requires the ability to describe hypergraphs. Below is a summary of the nature of a need for Jog Modeling Language (JogML) in order to further this research.

Why Yet Another Graph Language

The OpenStreetMap (OSM) dataset is an open source competitor to Google Maps and provides direct access to geospatial vector data via its planet.osm data file. The OpenStreetMap Markup Language, denoted by the .osm file extension, is a spatial implementation of Extensible Markup Language (XML) and provides a convenient way to define geographic types, especially roadways, and assign attributes to them. As in other vector file formats, nodes are given in a coordinate system, and edges are implied by the inclusion of multiple nodes within a shape. In OSM, road shapes are linestrings, which is denoted in OSM as Ways.

Although this street map data is well laid out for rendering maps and adequate for data transfer online, OSM requires multiple passes to recreate the network topology of the roadway system itself and is somewhat bloated for the purposes of RoadMapHash. Although a graph is implied by the data shapes in OSM, a graph-specific data language can reduce the overhead in parsing and recreating the graph structure by edges directly. Several such graph data languages exist. The most prominent is Graph eXchange Language (GXL) described in [9] and [10]. Despite the prevalence of GXL, Graph Modeling Language (GML), created by Michael Himsolt [11, 12] provides an excellent base for redefining the OpenStreetMap dataset for this research. Below is an explanation why this is.

Although GXL avails itself to existing XML parsers, GXL is unnecessarily general and verbose for this research. In particular, one goal of the JogML is to limit the value space of the terminals used in order to minimize the size of the serialized data files. Additionally, community support for a proposed file type is not necessary because this research provides a JogML parser, compiler, serializer, and deserializer. Finally, although GML does not provide a way to describe hypergraphs, it can be readily extended to do so. Hypergraphs are necessary here because the research makes use of a unique data structure called a Jog, consisting of three nodes. These three nodes comprise a one-way journey from a Start Node, to a Central Node, and to an End Node (see Figure 2 for an example of three Jogs over a sample road network). The three-node structure of a Jog is necessary for measuring road curvature, which must be computed for comparison with inertially sensed movement. Curvature is thus central to this research, and a data language that directly describes curvature during graph traversal will significantly reduce computation during data pre-processing. By redefining a road map as a set of Jogs, the graph becomes a hypergraph. Each Jog describes a subgraph consisting of a roadway node connected to two other roadway nodes, albeit in sequence. Hence the need to extend GML to support the Jog data structure.

JogML Grammar Specification and Example

The grammar specification of JogML is provided in Figure 6. Of particular note, Jogs are specified with a start, center, and end node, and each node optionally includes a list of Jog Starts, Jog Centers, or Jog Ends. This is due to the fact that a node may be member of one or more Jogs as its start, center, or end, but perhaps at the exclusion of the other(s). In order to allow for a reduction in storage space, these lists are made individually optional. All other attributes are required and sufficient. The Kleene cross (+) is also used to indicate the presence of one or more of the symbol preceding it. Additionally, the question mark (?) indicates an optional presence of the preceding symbol. That is, the symbol before ? may appear zero or one time.

Figure 7 provides a minimal example of a partial roadmap described with JogML. This OpenStreetMap data describes a footpath twenty meters east of Greene Hall on the URI campus. The Jogs and Nodes in this example replicate the topology of Figure 2. They do not completely describe the topology among those nodes and are meant only to be descriptive of JogML generally.

JogML Parse

As alluded to above, this work includes a recursive-descent LL(1) parser for the JogML format [8]. This simple hand-written parser uses the C++ extraction operator `>>` of the standard template library object `std::stringstream` as a lexer of white space delimited tokens. There is a one token buffer kept in a string variable "word", and although the parser style is recursive-descent, the JogML format never requires that recursion reach a depth greater than zero.

2.4.3 JogBin Data Format

In addition to JogML, this research produced the JogBin binary data format, whose specification is designed to make use of memory mapped files. By using the Unix-based mmap, or a similar system-specific memory interface, the binary data can be written and accessed as though it were virtual memory. Because our data is graph-like we can preserve the bidirectional connections between Nodes and Jogs by representing their edges, not as identification numbers, but as relative memory addresses. By retaining the connections within the binary data itself, the file can be used in-place as a complete data context. All preliminary computations are intrinsic to the file itself.

Issues with JogBin

In order to use relative memory addresses as a means to recreate the graph relationships in the data context, the machine writing the data must have the same data type sizes as the machine reading the data. Additionally, endianness may be a problem for raw binary reads and writes on different machines. Although there are some techniques to obviate these problems, the current implementation in this research only accounts for one architecture. Additionally, the version code in the file header is intended as a means to uniquely identify binary configurations, as well as version differences. Also, there is currently no system in place to keep the JogBin memory-mapped file threadsafe. Compare this with the implementation as used in MongoDB where the *mmap*ed files are sequentially locked for individual collections.

The three JogBin tables, Tables 1, 2, and 3, specify the three portions of a JogBin file. The data is first converted into data structures that have been optimized for 64-bit processor architecture [13], and then they are memory copied to their respective file locations in the memory-mapped file.

Table 1: JogBin Header Structure

Type	Data	Size in Bytes
uint64_t	Version code, specifies platform sizes	8
uint64_t	Total file size in bytes	8
long	Number of Nodes in the file	4
long	Number of Jogs in the file	4
float	Minimum Longitude of Nodes in the file	4

Table 2: JogBin Node Structure

Type	Data	Bytes
long long	OSM Node Identification Number	8
double	Latitude	8
double	Longitude	8

Table 3: JogBin Jog Structure

Type	Data	Bytes
long	ID Specific to JogML	4
float	Curvature, i.e. 1 / Turn Radius	4
float	Distance Between Start Node and Center Node, meters	4
float	Distance Between Center Node and End Node, meters	4
size_t	File Offset to Start Node	8
size_t	File Offset to Center Node	8
size_t	File Offset to End Node	8

2.4.4 JogML Improvement Considerations
MapReduce and Distributed Computation

A major advantage to serializing the data in the JogBin format is the compression that it affords. Much like Google's Protocol Buffer, JogBin can be transferred over a network much more quickly without losing the structure necessary for follow-on computation [14]. That is, JogBin is an excellent transfer and archival format without having the expense of requiring additional processing to use it. It is thus very well-suited to be used in a MapReduce distributed execution environment.

Database Engine

Currently the implementation of RoadMapHash uses C++ standard library containers, especially the `std::vector` and `std::map`. However, with data serialized in the JogBin format, the data could be traversed in-situ without conversion, and thus duplication, back into the original data types. With similar considerations in mind, database developers such as those at MongoDB have similarly employed memory mapped files have as a basis for their database engines [15].

Language Conclusions

The RoadMapHash algorithm has substantially benefited from the improvements of analysis from a Programming Language perspective. The JogML and JogBin formats advanced the RoadMapHash from being a series of scripts to forming the basis of a distributed graph database engine. The specification of JogML allows for a text-based serialization of the RoadMapHash data structures, but more importantly, it permitted the creation of JogBin, a binary datatype that can be used as a graph-database. Finally, the added benefit of data compression and fast serialization makes a distributed computation environment for RoadMapHash more effective.

2.5 Training Phase: RoadMapHash Algorithm

RoadMapHash is the first component of the RoadMapLocate system. It is our most unique contribution to the field of odometry-based localization. Just as in classic geometric hashing, the goal of RoadMapHash is to attempt to locate a shape within a larger one. However, the geometric hashing method used in this research does have two distinct differences.

First, unlike in classical geometric hashing, the query, or test shape, need not be scale invariant because the scale of the map is the same as the scale of the odometric sensors. This alleviates the combinatorial cost of projecting each node's position into the bases defined by the pairs of its k-nearest neighbors' coordinate systems. Instead, the coordinate system used to describe features can be the ego reference frame of the vehicle, with the center of the rear axle serving as the origin. Features can thus be described in terms of its hypothetical movements along a path, i.e. incremental changes in distance and direction.

Second, the features we seek to find in the hash table are not coordinates of nodes at all. This is because most nodes are distributed arbitrarily within a road section, and only describe along a continuous road space in the real world. As described in [16] as a "distinctive place," the localization framework must at some point convert continuous measurements into discrete features. Here the features we extract are high curvature areas. This leads to yet another issue: distinctiveness. Many high-curvature areas are similar.

Take for instance a traditional 90° right turn. On most US highways this maneuver will have a radius of curvature of approximately 5 meters and require a distance of approximately 8 meters to complete. If the feature were constructed from these two data, the feature's entry in the resulting geometric hash table would point to numerous similar right turn in the search space. This would not afford sufficient distinctiveness to separate the measured phenomenon from similar features in the map.

Fortunately, there is more data that can be used to build a suitable feature. Because the vehicle is measuring path distance, and inter-turn distance can be calculated by projecting the geographic coordinates to a flat earth model, we can link associate pairs of turns as distinct features.

The resulting turn-to-turn features, which we have defined as TurnEvents above, have two useful properties. First they more adequately distinguish similar road phenomena, and thus they reduce the number of collisions in the hash table. Second, because included in each feature itself is the distance from another, moving from TurnEvent to TurnEvent forms a near [5] transitive closure on the Ways in the map data. This fact will be used to threshold the likelihood that a sensed TurnEvent corresponds with a prior hypothesis.

2.5.1 Description

The RoadMapHash algorithm proceeds as follows.

Find Sharp Jogs Every Turn consists of at least one Sharp Jog. A Sharp Jog is a jog with a curvature above a specified threshold.

Find Turn Entry Jogs Every Turn has an Entry Jog and an Exit Jog. The entry and exit could be the same Jog, but not for all Turns. Consider a Turn that consists of a sharp curve going into the turn, followed by an intermediate curve of lesser but nonzero curvature, and finished with a sharp curve at the end. Such a Turn comprises several Jogs and must be identified with an approach that allows for multi-Jog Turns.

Find Turn Exit Jogs Starting with a Turn Entry as described above, Turns

[5] Dead ends and u-turns are not navigable by design decision.

proceed through a series of adjacent Jogs until they reach a Turn Exit Jog, which is the last Jog of high curvature in the Turn. The goal in this step is to identify where these Turn Exit Jogs occur. This process produces completed Turns which are a combination of Turn Exit Jogs and Turn Entry Jogs, as well as the array of intermediate Jogs between them.

Set Turn Origins As mentioned in the description above, a Turn on its own is difficult to distinguish in the real world. The degree of the turn and the distance traveled to complete the turn are the most useful, and possibly only, features that can be derived from the turn. However, the combination of a turn preceded by another turn offers a much richer set of data that can be both identified in a roadmap and sensed in the real world. In particular, the change in direction from the last turn, the distance traveled to the present turn, and the change in direction in the present turn can provide enough data for a geometric hash key to be specific but not result in an overly dense table. It is thus necessary to associate each Turn with an origin turn in order to create a Turn Event. This is done by a depth first traversal from the exit turn of a Turn T_A to any and all entry Turns $T_{\{Entry\}}$, within a search limit set by a configuration parameter. See Figure 11 for an example of a Turn with several origin Turns.

Create TurnEvents TurnEvents are for the basis of the geometric hash map key and are combinations of a Turn and a Turn Origin. Scaled rounding [17] is applied to map-based distance and angle displacements in order to create set of keys which are robust to the uncertainty of real world measurement. The scale factor used here relates directly to the error variance used in the motion model.

Create Hash Table With TurnEvents The geometric hash table for a given area must be updated with the TurnEvents occurring in that area. The key is the string-encoded Turn pair phenomenon found in the map data. The value is the location (or optionally, several locations) where the phenomenon occurred.

2.5.2 Complexity Analysis

Using the Random Access Machine model of computation, we calculate up to a constant factor the worst case runtime of the RoadMapHash algorithm with only the number of Jogs in the roadmap as input size. Although this calculation relies on Jog-based data, one can also derive an upper bound on the number of Jogs within a geographic area by using only the number of Nodes. That derivation, followed by a brief time-complexity analysis of the steps of the RoadMapHash algorithm follow.

Number of Jogs in a Digital Road Map

Road map data describes a graph whose points and edges reside in a plane. However, for a graph G to be *planar* not only must each node reside in the plane, but no edge may intersect with another edge. Luckily, we are given exactly that constraint as a stipulation in the OpenStreetMaps (OSM) specification [cite http://wiki.openstreetmap.org/wiki/Node#NodesonWays]. Because OSM street data is planar, we can invoke Euler's Formula described in Theorem 2.5.1 in order to provide an upper bound on the number of nodes and edges. This theorem and a proof are given in [18, pg. 22].

Theorem 2.5.1 (Euler's Formula)**.** Let G be a connected planar graph with n vertices, m edges and f faces. Then $n - m + f = 2$.

From Euler's Formula, Kleitman in [19] proves a conclusion used for graph coloring that we can apply here. That conclusion and Kleitman's proof are given as Lemma 2.5.2.

Lemma 2.5.2. A planar graph on v vertices can have at most $3v - 6$ edges and average degree strictly less than 6.

Proof. Each region defined by a drawing of G in the plane is bounded by a cycle

Algorithm 1 RoadMapHash

```
function RMHash($A_{jog}$)
    Let $A_{SJ}$ be a new dynamic array of Sharp Jogs
    for each Jog $J$ in $A_{jog}$ do                                ▷ Find Sharp Jogs
        if J curvature $C > C_{thresh}$ then
            $A_{SJ} += J$
        end if
    end for
    Let $S_{JEnt}$ be an unordered set of Turn Entry Jogs
    for each Sharp Jog $J$ in $A_{SJ}$ do                           ▷ Find Turn Entry Jogs
        if Sharp Jog Not Already Explored then
            while DFS Each Upstream Jog $J_i = J_{i-1} \rightarrow J_{prev}$ do
                if J curvature $C(J_{prev}) < C_{thresh}$ or $C(J) * C(J_{prev}) < 0$ then
                    $S_{JEnt} += J_{prev}$
                end if
            end while
        end if
    end for
    Let $S_T$ be an unordered set of Turns
    for each Turn Entry Jog $J$ in $S_{JEnt}$ do                    ▷ Find Turn Exit Jogs, Turns
        while DFS Each Downstream Jog $J_i = J_{i-1} \rightarrow J_{next}$ do
            if J curvature $C < C_{thresh}$ or $C(J) * C(J_{next}) < 0$ then
                $S_T += T(\{J : J_{next}\})$
            end if
        end while
    end for
    for each Turn $T$ in $S_T$ do                                   ▷ Set Turn Origins
        while DFS Each Jog $J_{prev}$ Upstream from $T \rightarrow J_{entry}$ do
            if $J_{prev}$ is a $J_{exit}$ for any Turn $T_i$ then
                $T \rightarrow TurnOrigins += T_i$
            end if
        end while
    end for
    Let $S_{TE}$ be a set of TurnEvents
    for each Turn $T_i$ in $S_T$ do                                 ▷ Create Turn Events
        for each Turn $T_j$ in $T_i \rightarrow TurnOrigins$ do $S_{TE} += TE(T_i, T_j)$
        end for
    end for
    Let $M$ be a map of type <string, vector<pair<float, float>>>
    for each TurnEvent $TE$ in $S_{TE}$ do                          ▷ Create Hash Table
        Insert $TE \rightarrow Signature, Location$ in $M$
    end for
end function
```

of G. If that cycle is not a triangle, we can add an edge between two opposite vertices and increase the number of edges.

We conclude then that a graph G on v vertices with the most edges will have triangles for all its faces.

Then each face has three edges on its boundary, and the number of edge-face pairs with the edge bounding the face will be $3f$.

But each edge bounds 2 faces, so that the number of edge-face pairs with the edge bounding the face will also be $2e$.

We deduce then that $f = 2e/3$, so that in this case we can write Euler's formula as

$$v - e + \frac{2e}{3} = 2$$
$$3v = e + 6$$

We notice also that the number of edge-vertex pairs with the vertex in the edge, is $2e$ and is also the sum of the degrees of all the vertices.

This tells us that the sum of the degrees of the vertices of any edge maximal planar graph on v vertices obeys:

The sum of the degrees of the vertices of G is $2e$ or $6v - 12$.

The average degree of a vertex of G is therefore, $6 - 12/v$. □

With this important observation as a basis, we can then draw the following conclusion about the greatest number of Jogs in a Roadmap.

Theorem 2.5.3 (Greatest Number of Jogs in Roadmap)**.** Let $|J|$ be the cardinality of the set of Jogs in a roadmap G and $|N|$ be the number of Nodes in G. Then

$$|J| < 36|N|.$$

Proof. Choose any node n in the roadmap G. Let N_{in} be the set of nodes adjacent to n which define edges in-bound to n, and let N_{out} be the set nodes with edges out-bound from n. By Definition 2.2 of a Jog, the number of Jogs central to n is the size of the set of ordered pairs $(N_{in} \times N_{out}) - C$, where $C = \{(n_{in}, n_{out}) | n_{in} = n_{out}\}$, and C is then equal to the intersection $N_{in} \cap N_{out}$. So we have that

$$\begin{aligned} |J| &= |(N_{in} \times N_{out}) \backslash (N_{in} \cap N_{out})| && (1) \\ &= |(N_{in} \times N_{out})| \backslash |(N_{in} \cap N_{out})| && (2) \\ &<= |(N_{in} \times N_{out})| && (3) \\ &<= |E|^2, \text{ where E is the set of edges in G} && (4) \\ &< 6^2|N| = 36|N| \text{ by Lemma 2.5.2} && (5) \end{aligned}$$

□

Time Complexity of RoadMapHash

Find Sharp Jogs As shown in lines 3 through 7 of Algorithm 2.5.1, the search for Sharp Jogs is linear with respect to the number of Jogs. As detailed in Subsection 2.5.2, the number of Jogs is linearly proportional to the number of Nodes in the map data. So finding Sharp Jogs is $O(N_{Nodes})$ in time.

Find Turn Entry Jogs Lines 9 through 17 of Algorithm 2.5.1 show that for each Sharp Jog, a depth first search on the space of Jogs "upstream", or in the direction opposite the flow of traffic, from the Sharp Jog. In practice this search is limited to a fixed distance and we can conclude that the time required is of the order $O(N_{Jogs} \times 1) = O(N_{Nodes}) = O(N)$; however, as in the above if we allowed the search to continue without a distance constraint,this step would require $O(N_{Jogs} \times N_{Jogs}) = O(N_{Nodes} \times N_{Nodes}) = O(N^2)$ time. [6]

Find Turn Exit Jogs Lines 19 through 25 of Algorithm 2.5.1 show a phase of RoadMapHash that is nearly identical to the previous step. Here, for each Turn Entry Jog, a depth-first search is performed on the space of Jogs "downstream," or in the direction of the flow of traffic away from the Turn Entry Jog. Again, this search is limited to a fixed distance and we can conclude that the time required is of the order $O(N)$; however, as in the above, if we allowed the search to continue without a distance constraint, this step would require $O(N^2)$ time. At this step we will observe that there are no more than N^2_{Jogs} Jog pairs, and thus the number of Turns has an upper bound of $O(N^2)$ as well.

Set Turn Origins As lines 26 through 32 in Algorithm 2.5.1 indicate, Setting Turn Origins is a process nearly identical to Finding Turn Entry Jogs, and its asymptotic cost is proportional. So we have for each Turn (of which we have $O(N^2)$), we must search the space of Jogs once more. This step thus requires $O(N^3)$ time with input size N given by the number of Nodes in the roadmap data. Note that this is consistent with the complexity analysis provided by Wolfson and Rigoutsos in [21], where regarding classical geometric hashing the authors claim that

> In general, if
>
> - The database contains M known models, each comprising n features,
> - The scene during recognition contains S features, and
> - c features are needed to form a basis,

[6]We note for completeness that there do exist $O(log(N))$ algorithms for DFS traversal of planar graphs [20], but we remind the reader that the hypergraph formed by Jogs is not itself planar. Consider for a a simple counterexample the three Jogs in Figure 2. Note that Jog 2 crosses both Jog 1 and Jog 3, and in general, Jogs may cross each other just as the set of drivable routes over a roadmap must also cross.

then the time complexity of the preprocessing phase is $O(Mn^{c+1})$. [21, pg. 15]

In this research, the number of Turn features we use is 2, and our preprocessing phase is dominated by a $O(N^3)$ operation.

Create TurnEvents As seen in lines 34 through 37 of Algorithm 2.5.1, creating "TurnEvents", or the key-value pairs to be inserted in the hash table is an operation propotional to the number of Turn-TurnOrigin combinations found. This is bounded above by $O(N^3)$, and because inserting into an ordered set is a $O(1)$ operation, this operation is also bounded by $O(N^3)$ for N Nodes in the roadmap[7].

Create Hash Table With TurnEvents Inserting into a hash table is a $O(1)$ operation and this is also linear over the set of $O(N^3)$ TurnEvents.

We remind the reader that in considering the runtime of RoadMapHash as a function of the number of input nodes, we do constrain the search for Turns and TurnEvents geographically. This is intuitive because a real-world turn event could not span across continents, for instance. Similarly, we constrain the search of Turn pairs (and thus TurnEvents) to be within the extent of a configured search radius, nominally 10 kilometers. Thus the runtime of the entire system becomes linear with respect to the total number of these search extents, as defined by the geographical area we wish to cover. See Figure 12 for an illustration of this distribution.

2.5.3 Distributed RoadMapHash

The RoadMapHash algorithm has been parallelized to the extent that it makes nearly maximum use of eight virtual cores. However, the vast data processing

[7]Although an ordered set is suitable for production, in our test implementation we use an ordered `std::map` from the C++ Standard Template Library for debugging purposes.

demands of the RoadMapHash system exceed the computational capacity of a single consumer PC. In order to further parallelize the RoadMapHash system, it must be distributed across multiple computational units. This section will discuss how the RoadMapHash system has now scaled horizontally as well as vertically.

Phases of Parallelization

Below is a brief summary of the RoadMapHash data processing algorithm, which can be broken down into four key phases.

1. Raw Data Download. The unprocessed data in the form of OpenStreetMaps's version of XML, known as OSM, is downloaded from public servers. These servers also provide some data filtering, such as limiting the type data only to larger, arterial roads, for instance.

2. Jog Assembly from Nodes and Ways. The unprocessed data is a set of nodes and a set of linestrings, known in OSM as Ways. In order for the curvature based RoadMapHash algorithm to quickly traverse the roadmap graph in search of curvatures, this data is reassembled into three-node sets called Jogs.

3. Turn Search and Turn Pair Matching. Once the set of Jogs is available, RoadMapHash can determine where in the road network turns can be sensed by an inertial measurement unit and a speedometer. These turn features need not simply be Jogs with high curvature; rather, they must be uniquely identifiable with sensors. Thus locating "Turns" is somewhat more complicated and requires traversing the Jog graph extensively. Additionally, each Turn is paired with another Turn by means of calculating the distance and change in heading between the two. This is done in order to further distinguish Turn features to spread out the lookup table.

4. Hash Table Updating. When the Turn features are discovered, they each provide a unique turn signature and a set of coordinates to the location of the turn. This key-value pair is entered into a MongoDB database by means of batch uploading, and in turn upserting, JSON documents over the Internet. This is the “Reduce” phase of the MapReduce process.

Distributed Implementation

Before the implementation of a distributed architecture, RoadMapHash had been implemented in C++ without a command line interface. Raw data was acquired before execution and processed hash tables were uploaded to the database one at a time with a manually executed script. Additionally, the actual data processing, comprising the second and third phases of the RoadMapHash algorithm, was performed by one machine on the same set of data from start to finish. This following implementation provided opportunities for improvement using distributed parallelization. Note that the parallel decomposition relies on two variables: the number of servers as well as the number of map segments (and thus client threads).

Overall Workflow

1. Divide the map area into X equal sized rectangloids (not quite rectangles because they are segments of the surface of an ellipsoid).

2. Provision N RoadMapHash worker servers.

3. Allocate the X segments to the N servers to do work.

Technologies Used

- C++. As mentioned above, the RoadMapHash algorithm itself is implemented in C++ and uses LLVM C++ standard library threads (`std::thread`) in order to parallelize work within each task.

- Go. The workers serve a web service written in Golang. There are REST endpoints corresponding to the functions of the RoadMapHash algorithm. There is not currently seamless integration between the RoadMapHash algorithm and this web service. Rather, the binary 'rmhash' utility is called as an external process and the file system is used to poll the completion of each file.

- Python. The distributed worker driver is written in Python with the Requests library used to perform HTTP requests on the servers. The workflow described above is implemented in Python and extends the 'threading.Thread' class in order to achieve parallel computation there.

- Docker. This project made use of Docker containers in order to deploy the web application. A future goal of the research will be to orchestrate these containerized services with Kubernetes.

- Cloud Resources. This project made use of virtual compute resources from Microsoft Azure and Amazon Web Services.

Results

The results in Figure 13 show that the greatest speedup gained was approximately 2, when four map sections were processed. This is the result of a 1:1 map section-to-server ratio, and the fact that each server had to download map data. This bottleneck at the data download process created a linear time overhead in the number of map sections, probably due to the slow down caused by multiple HTTP requests and file handling. In general, however, the distributed architecture shows promise. With 4 servers and 4 map sections, the distributed RoadMapHash algorithm achieved a 1.90x speedup over a single machine working on a single map section of the same area. This is consistent with the possibility that there is an optimal coverage area that will balance the speedup gained in parallel processing maps against the slow down caused by multiple data downloads and file operations.

Directions for Further Improvements

As mentioned above, the integration of the RoadMapHash algorithm with the web API could be improved by making direct calls on a linked library as opposed to making system calls to a binary executable, and then polling for completion. Additionally, as mentioned in the results, the map data could be downloaded in segments of optimal size because the downloads against the OSM server took a significant portion of the within-worker time. Finally, the current system does not yet implement the "Reduce" phase of the MapReduce workflow. That is, the data is not yet merged back into a central data source. The scheme by which this data will be merged must also be carefully considered. Because lookup tables for each sub-region of the map may adequately describe a search space, each could be independently useful to delineate the data result between service regions.

2.6 Tracking Phase: RoadMapTrack

The RoadMapTrack algorithm is the second subcomponent of the overall RoadMapLocate system. Its purpose is limited and is designed operate on an embedded system with as few features as necessary to record sensor data and perform minor arithmetic calculations for final computation by another device. Ultimately, RoadMapTrack will provide estimates of path displacement ΔU_t, change in heading $\Delta\theta_t$, instantaneous radius of curvature R_{curve_t}, and a timestamp τ_t, for on-line or off-line processing by the RoadMapFilter algorithm.

See in Algorithm 2 that RoadMap Track serves as a primary arithmetic stage of sensor data processing. This is by design. Future implementations of RoadMap-Track could included features such as run-length encoding for data records, and possible integration with an IMU or GPS; however, in general we intend to restrict the capabilities of RoadMapTrack so that it can operate on a stable, single purpose platform. Also, see that Line 6 of the algorithm requires the constant b be defined. This necessitates that the vehicle's track width be known ahead of time in order to use the data supplied by RoadMapTrack algorithm as an intermediate virtual sensor. Finally note that in Line 7 of Algorithm 2, we compute the Radius of Curvature by division. This requires checking that the change in heading $\Delta\theta_t$ be non-zero, and otherwise returning infinity.

Algorithm 2 RoadMapTrack

```
1: function RMTrack($\dot{U_{Lt}}, \dot{U_{Rt}}, \tau_t$)
2:     Let $\tau = \tau_t - \tau_{t-1}$                                   ▷ Time Since Update
3:     Let $\Delta U_{Lt} = \dot{U_{Lt}} * \tau$                          ▷ Left Wheel Displacement
4:     Let $\Delta U_{Rt} = \dot{U_{Rt}} * \tau$                          ▷ Right Wheel Displacement
5:     Let $\Delta U_t = (\Delta U_{tL} + \Delta U_{tR})/2$               ▷ Path Displacement
6:     Let $\Delta\theta_t = (\Delta U_{tL} - \Delta U_{tR}) * b$         ▷ Heading Displacement
7:     Let $R_{curve_t} = (\Delta U_t/\Delta\theta_t) - (b/2)$            ▷ Radius of Curvature
       return $(\Delta U_t, \Delta\theta_t, R_{curve_t}, \tau_t)$
8: end function
```

2.7 Localization Phase: RoadMapFilter Algorithm

In order to accurately estimate the pose of the vehicle[8] we must review some of the arithmetic performed by the RoadMapTrack subcomponent.

2.7.1 Motion Model

Differential odometry is commonly used in two-wheeled robots and vehicles with differential drive in general. Here we use the differential drive model described in [22, pg. 19] coupled with the odometry motion model described in [23, pg. 95]. State updates from the wheel speed sensors are interpreted as changes in position of the left and right wheels respectively. In the following description of

[8]We will refer to a "pose" of the vehicle as a hypothesis of its relative translation and rotation from the last feature event.

our model we refer to the "pose" of the vehicle, Equation 6 as a three dimensional vector corresponding to its horizontal translation, vertical translation, and change in bearing from *a previously known pose* X_{t_k-1}. This distinction is important for further discussion because we uniquely expand on the need for a single vehicle-intrinsic coordinate frame by updating this reference frame at each measurement event k.

$$X_{t_k} = \begin{bmatrix} \theta_{t_k} \\ x_{t_k} \\ y_{t_k} \end{bmatrix} \tag{6}$$

We now look at the motion model which is similar to those used in [22] and [24]. The differential odometry model presented by Borenstein *et al.* describes a vehicle whose left drive wheel moves Δ_{U_L} and whose right drive wheel moves Δ_{U_R}. Borenstein's model describes x, y, and θ displacement as a function of these updates.

As in [22], the vector represented by U_{t_k} refers to a the *control vector*. Although we discuss the control vector to remain consistent with the terminology used to describe control theory in general, we point out that the measurements amounting to this control are taken after the fact in our input-free system. As in [22], we similarly use the convention of a positive ΔD_{t_k} to refer to forward motion and a positive $\Delta\theta_{t_k}$ to refer to counter-clockwise rotation. One distinction to be made is that we do not refer to the subscript t to be a global time elapsed since the process began; rather we intend to refer to the time t_k as the time instant since measurement event k, which itself refers to the time of an update event as prompted by recognizing a road feature. However, because t_k is cumbersome, from here forward, we will refer merely to the time t.

In considering the differential drive, we describe our update vector as Equation 7.

$$\Delta D_t = (\Delta D_{tL} + \Delta D_{tR})/2\Delta_{\theta_t} = (\Delta D_{tL} - \Delta D_{tR})/b \tag{7}$$

These updates in path and heading change relate to our state vector with the differential drive system described in System 8 [23], [24].

$$F = \begin{cases} f(\theta_{t-1}) & = \theta_t = \theta_{t-1} + \Delta\theta_t \\ f(x_{t-1}) & = x_t = x_{t-1} + \Delta D_t \cos\theta_t \\ f(y_t) & = y_t = y_{t-1} + \Delta D_t \sin\theta_t \end{cases} \tag{8}$$

Finally, the system state vector can thus be written as $X_t = [x_t, y_t, \theta_t]^\tau$, the update vector as $\Delta U_t = [\Delta D_t, \Delta_{\theta_t}]^\tau$, and the function vector $\mathbf{f}(\mathbf{X}) = [\mathbf{f_x}, \mathbf{f_y}, \mathbf{f}_\theta]^\tau$.

2.7.2 Uncertainty

In Equation 7 we give the relative orientation given by difference in change in distance of the left wheel ΔD_{tL} and change in distance of the right ΔD_{tR}, where b is the distance between the left rear wheel and the right rear wheel [22][9]. However, because the wheel speed sensors are error prone, and the effects of odometric error are cumulative, we model the true control vector as System 9.

$$\Delta \hat{D}_t = \Delta D_t + \epsilon_U \tag{9}$$

$$\hat{\Delta}_{\theta_t} = \Delta_{\theta_t} + \epsilon_\theta \tag{10}$$

In the literature an error in the observed or commanded change in heading Δ_θ is occasionally modeled as a triangular distribution [23]. This model is suitable for skidsteer vehicles and in robots with actual differential drivetrains, where the data generating process is a control on the degree of z axis twist, and ultimately

[9]The distance between the two rear wheels is known as the track width.

produces an “arc” of uncertainty. In [25] and [23], once the vehicle proceeds forward after a change in rotation (or some sequence of similar infinitesimal movements) the error in rotation should be considered from the perspective of z axis rotations followed by forward movement.

However, here the data generating process is not control on the vehicle’s differential drivetrain, rather, we passively collect wheel speed sensor data gathered from both rear wheels. The error for this data needs to be modeled accordingly, and we plan to implement an exponential parametric error system, describing uncertainty in translation and rotation, as in [26]. In [26] the authors use a learned parametric exponential model to test goodness-of-fit between range measurements and model values.

2.7.3 Turn State Machine

Figure 16 shows the rule set used to determine whether a vehicle traveling on a road of known curvature is in a turn or out of a turn. This finite state machine is designed to threshold two levels of curvature. One is for maintaining a turn state, when for example a vehicle travels a slightly curved portion of the road, but returns to a straightaway. The second is a threshold after which the vehicle is considered to be in a turn, which is nominally a turn radius of 15 meters. This two-phase turn state process allows for vehicles to move with variance in road curvature without overfitting too many turn events to the sequence of input.

2.8 RoadMapFilter

For the third phase of RoadMapLocate, we propose a sequential Monte Carlo estimation process similar to the particle filter. The system should best approach the ones specified by [27], [26], and [28] where estimates are probabilistic over a set of hypotheses, and in some cases, the measurement phase requires adjustment to the standard particle filter. Algorithm 3 sketches a possible method. Missing however, is the incorporation of assigning probabilistic weights to the hypotheses. As mentioned above in Motion Model, we suspect that the error models in [26] will serve as a guide for this. Additionally, because the virtual measurements from the hashtable may be multiple and very different, we further propose using the gating technique in [27]. This is reflected in line 16 of Algorithm 3.

Algorithm 3 RoadMapFilter

```
 1: function RoadMapFilter($\bar{X_0}, \sigma_X, u_t$)
 2:     Sample $X_0 \sim \mathcal{N}_{(\S,\dagger)}(\bar{X_0}, \sigma_X), \mathcal{U}_\theta(-\pi, \pi)$            ▷ Initialize particles
 3:     while $u_{t+1} \neq \emptyset$ do                                      ▷ Continuously Update
 4:         for $i = 1$ to $Len(X_t)$ do                                   ▷ Loop over Hypotheses
 5:             $x_i = x_i + u_t + \epsilon_x$                                ▷ Update Pose Hypotheses
 6:         end for
 7:         if Turn feature $k$ detected from $u_t$ then
 8:             $Z_t = RoadMapHash(k)$                                  ▷ Virtual Measurements
 9:             for $i = 1$ to $N$ do
10:                 Sample $x_i$ from $\{X_t\}$               ▷ Sample from $X_t$ with replacement
11:             end for
12:             for $j = 1$ to $Len(z_t)$ do                              ▷ Loop over Candidates
13:                 Add $z_j$ to $X_t$                                          ▷ New Hypothesis
14:                 for $i = 1$ to $Len(X_t)$ do                          ▷ Loop over Hypotheses
15:                     if $x_i \approx z_j$ then
16:                         $x_i = z_t$                                      ▷ Merge if Similar
17:                     end if
18:                 end for
19:             end for
20:         else
21:         end if
22:     end while
23: end function
```

List of References

[1] O. Members, "OGC® Standards and Supporting Documents," 2017. [Online]. Available: http://www.opengeospatial.org/standards

[2] J. Bruce, J. Wawerla, and R. Vaughan, "The SFU mountain dataset: Semi-structured woodland trails under changing environmental conditions," in *Proceedings of IEEE International Conference on Robotics and Automation 2015,*

Workshop on Visual Place Recognition in Changing Environments, Seattle, WA, USA, 2015.

[3] Advanced Navigation, "OBDII Odometer Reference Manual," Web, 2014. [Online]. Available: http://www.advancednavigation.com.au/sites/advancednavigation.com.au/files/obdii_odometer_reference_manual.pdf

[4] Wikipedia Contributors, "OBD-II PIDs," 2017. [Online]. Available: https://en.wikipedia.org/wiki/OBD-II_PIDs

[5] M. E. B. E. Najjar and P. Bonnifait, "Road Selection Using Multicriteria Fusion for the Road-Matching Problem," *Proceedings of the IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 279–291, June 2007.

[6] Neo4j Staff, "The Database Model Showdown: An RDBMS vs. Graph Comparison - Neo4j Graph Database," 2017. [Online]. Available: https://neo4j.com/blog/database-model-comparison/

[7] Facebook, "GraphQL," 2016. [Online]. Available: http://facebook.github.io/graphql/October2016/

[8] Wikipedia Contributors, "Data-centric programming language," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Data-centric_programming_language

[9] V. Engelmann, "www.Open-GraphTheory.org," 2017. [Online]. Available: http://www.open-graphtheory.org/gxlformat.html

[10] S. Mohammed and M. Bernard, *Graph File Formats*, 2004. [Online]. Available: http://www2.sta.uwi.edu/~mbernard/research_files/fileformats.pdf

[11] M. Himsolt, *GML: A Portable Graph File Format*. [Online]. Available: https://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf

[12] Wikipedia Contributors, "Graph modelling language," 2017. [Online]. Available: https://en.wikipedia.org/wiki/GraphModellingLanguage

[13] E. Raymond, "The Lost Art of C Structure Packing," 2017. [Online]. Available: http://www.catb.org/esr/structure-packing/

[14] Google, "Protocol Buffers — Google Developers," 2017. [Online]. Available: https://developers.google.com/protocol-buffers/

[15] L. Petit, "MongoDB WiredTiger: Why We Switched Back to MMap," 2015. [Online]. Available: https://tech.c-radar.com/2015/10/12/mongodb-wiredtiger-why-we-switched-back-to-mmap/

[16] B. J. Kuipers and Y. Byun, "A Qualitative Approach to Robot Exploration and Map-Learning," in *Workshop on Spatial Reasoning and Multi-Sensor Fusion*, St. Charles, IL, USA, Oct. 1987, pp. 390–404.

[17] Wikipedia Contributors, "Rounding," 2017.

[18] D. Jungnickel, *Graphs, Networks and Algorithms*, 4th ed. New York, NY, USA: Springer, 2014.

[19] D. Kleitman. "Planarity and Coloring." 2004. [Online]. Available: http://www-math.mit.edu/~djk/18.310/18.310F04/planarity_coloring.html

[20] T. Hagerup, "Planar Depth-First Search in O(log n) Parallel Time," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 678–704, 1990. [Online]. Available: https://doi.org/10.1137/0219047

[21] H. Wolfson and I. Rigoutsos, "Geometric Hashing: An Overview," *IEEE Computational Science and Engineering*, vol. 4, no. 4, pp. 10–21, 1997.

[22] J. Borenstein, H. Everett, and L. Feng, "Where am I? Sensors and Methods for Mobile Robot Positioning, Tech. Rep. 120, 1996.

[23] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*, ser. Intelligent Robotics and Autonomous Agents. Cambridge, Massachusetts, USA: The MIT Press, Aug. 2005. [Online]. Available: http://www.worldcat.org/isbn/0262201623

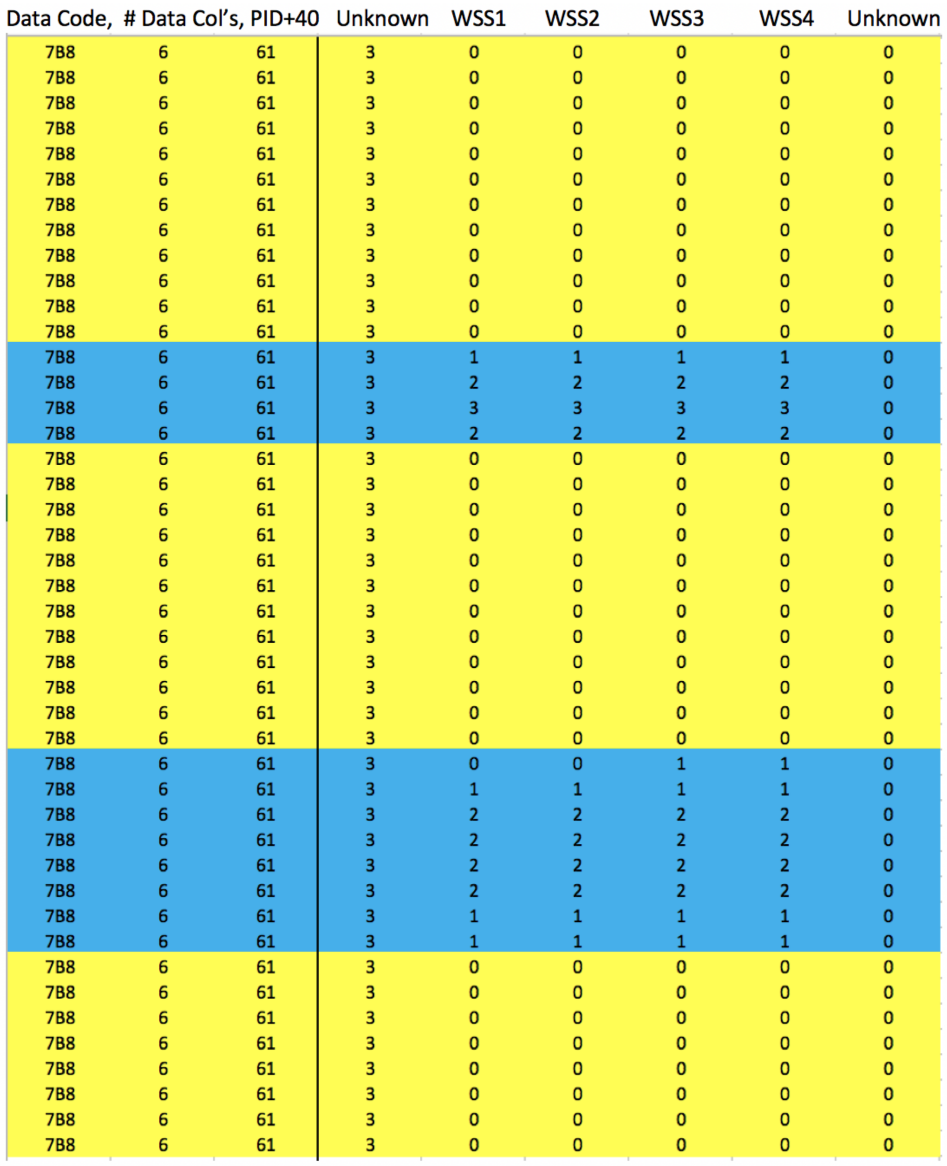
[24] E. Kiriy and M. Buehler, "Three-State Extended Kalman Filter for Mobile Robot Localization," Montreal, Que., Canada, Tech. Rep., 2002.

[25] K. T. Sutherland and W. B. Thompson, "Localizing in Unstructured Environments: Dealing With the Errors," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 6, pp. 740–754, Dec 1994.

[26] H. Badino, D. Huber, and T. Kanade, "Real-Time Topometric Localization," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA, May 2012, pp. 1635–1642.

[27] P. Jensfelt and S. Kristensen, "Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, Oct. 2001.

[28] R. Karlsson and F. Gustafsson, "The Future of Automotive Localization Algorithms: Available, Reliable, and Scalable Localization: Anywhere and Anytime," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 60–69, March 2017.



Data Code,	# Data Col's,	PID+40	Unknown	WSS1	WSS2	WSS3	WSS4	Unknown
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	1	1	1	1	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	3	3	3	3	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	1	1	0
7B8	6	61	3	1	1	1	1	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	2	2	2	2	0
7B8	6	61	3	1	1	1	1	0
7B8	6	61	3	1	1	1	1	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0
7B8	6	61	3	0	0	0	0	0

Figure 4. OBDII Parameter ID Discovered by Logging During Short Drive. Movement periods highlighted in blue.

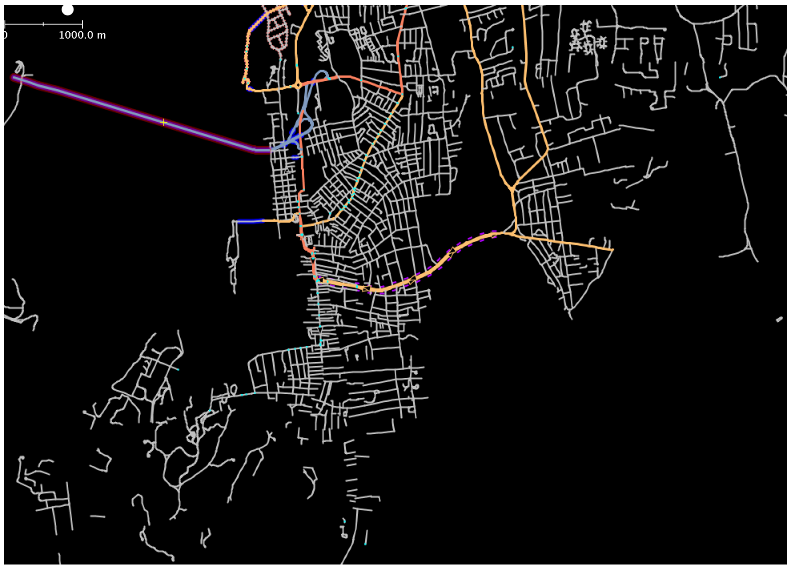
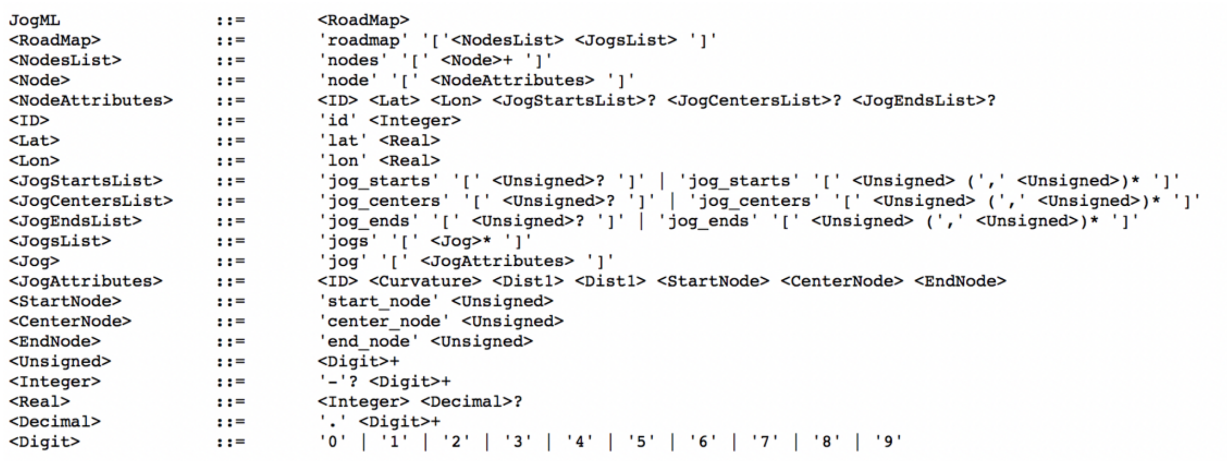
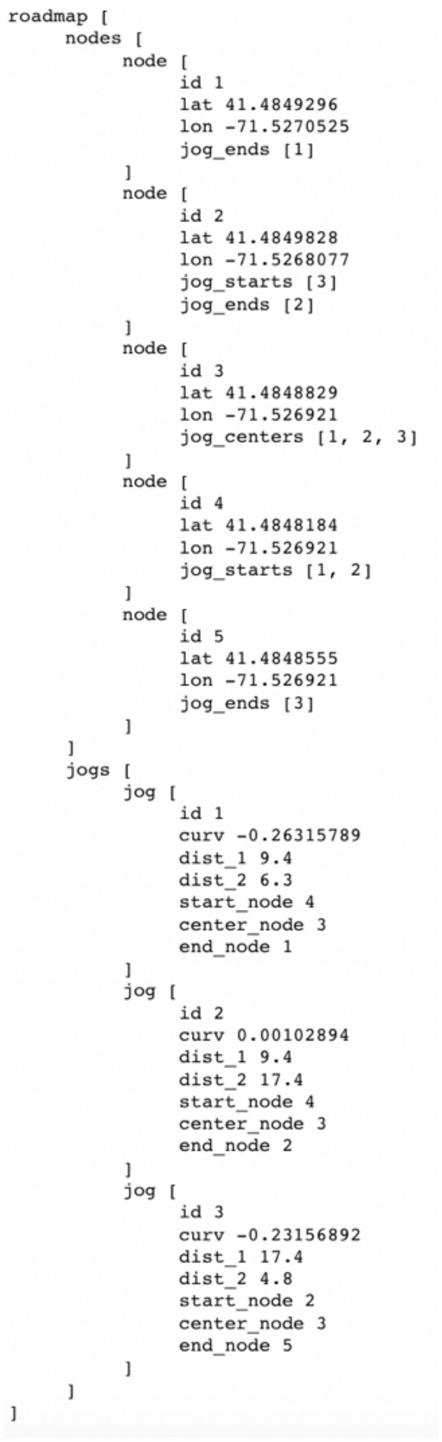


Figure 5. Example OSM Roads in Newport, RI. Image from JavaOpenStreetMaps (JOSM).



```
JogML                ::=       <RoadMap>
<RoadMap>            ::=       'roadmap' '['<NodesList> <JogsList> ']'
<NodesList>          ::=       'nodes' '[' <Node>+ ']'
<Node>               ::=       'node' '[' <NodeAttributes> ']'
<NodeAttributes>     ::=       <ID> <Lat> <Lon> <JogStartsList>? <JogCentersList>? <JogEndsList>?
<ID>                 ::=       'id' <Integer>
<Lat>                ::=       'lat' <Real>
<Lon>                ::=       'lon' <Real>
<JogStartsList>      ::=       'jog_starts' '[' <Unsigned>? ']' | 'jog_starts' '[' <Unsigned> (',' <Unsigned>)* ']'
<JogCentersList>     ::=       'jog_centers' '[' <Unsigned>? ']' | 'jog_centers' '[' <Unsigned> (',' <Unsigned>)* ']'
<JogEndsList>        ::=       'jog_ends' '[' <Unsigned>? ']' | 'jog_ends' '[' <Unsigned> (',' <Unsigned>)* ']'
<JogsList>           ::=       'jogs' '[' <Jog>* ']'
<Jog>                ::=       'jog' '[' <JogAttributes> ']'
<JogAttributes>      ::=       <ID> <Curvature> <Dist1> <Dist1> <StartNode> <CenterNode> <EndNode>
<StartNode>          ::=       'start_node' <Unsigned>
<CenterNode>         ::=       'center_node' <Unsigned>
<EndNode>            ::=       'end_node' <Unsigned>
<Unsigned>           ::=       <Digit>+
<Integer>            ::=       '-'? <Digit>+
<Real>               ::=       <Integer> <Decimal>?
<Decimal>            ::=       '.' <Digit>+
<Digit>              ::=       '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Figure 6. JogML Grammar in Extended Backus-Naur Form (EBNF)



```
roadmap [
     nodes [
          node [
               id 1
               lat 41.4849296
               lon -71.5270525
               jog_ends [1]
          ]
          node [
               id 2
               lat 41.4849828
               lon -71.5268077
               jog_starts [3]
               jog_ends [2]
          ]
          node [
               id 3
               lat 41.4848829
               lon -71.526921
               jog_centers [1, 2, 3]
          ]
          node [
               id 4
               lat 41.4848184
               lon -71.526921
               jog_starts [1, 2]
          ]
          node [
               id 5
               lat 41.4848555
               lon -71.526921
               jog_ends [3]
          ]
     ]
     jogs [
          jog [
               id 1
               curv -0.26315789
               dist_1 9.4
               dist_2 6.3
               start_node 4
               center_node 3
               end_node 1
          ]
          jog [
               id 2
               curv 0.00102894
               dist_1 9.4
               dist_2 17.4
               start_node 4
               center_node 3
               end_node 2
          ]
          jog [
               id 3
               curv -0.23156892
               dist_1 17.4
               dist_2 4.8
               start_node 2
               center_node 3
               end_node 5
          ]
     ]
]
```

Figure 7. JogML Example

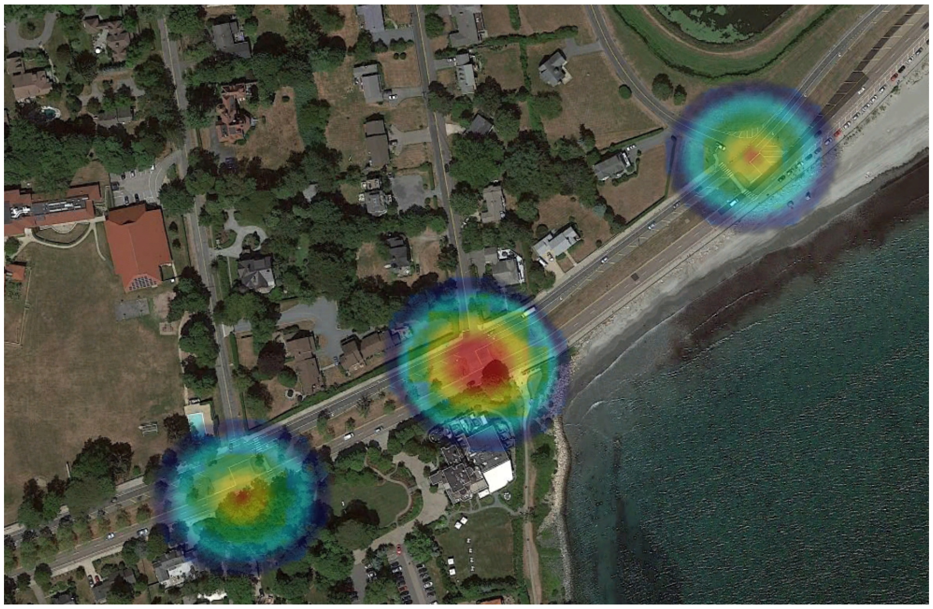


Figure 8. High Curvature Areas Highlighted. Google Image.

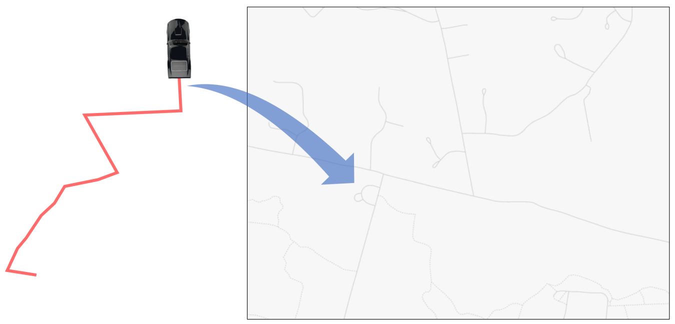


Figure 9. Path Segment Shape Identified, Map Data Known

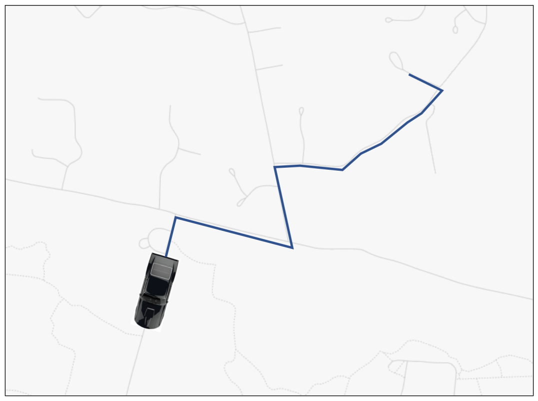


Figure 10. Path Segment is Matched to Road Map Data

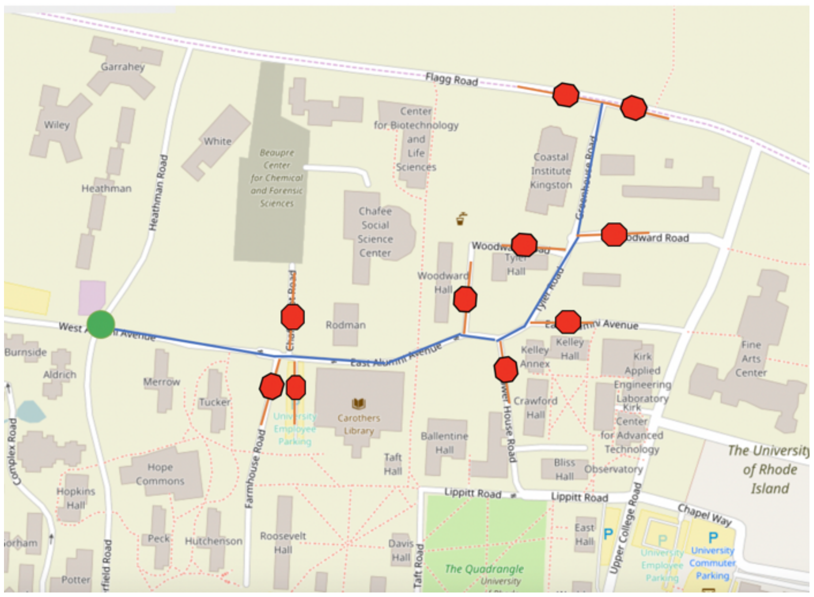


Figure 11. Several Turn Events Terminating at Red Turns, Each Originating from a Single Green Turn

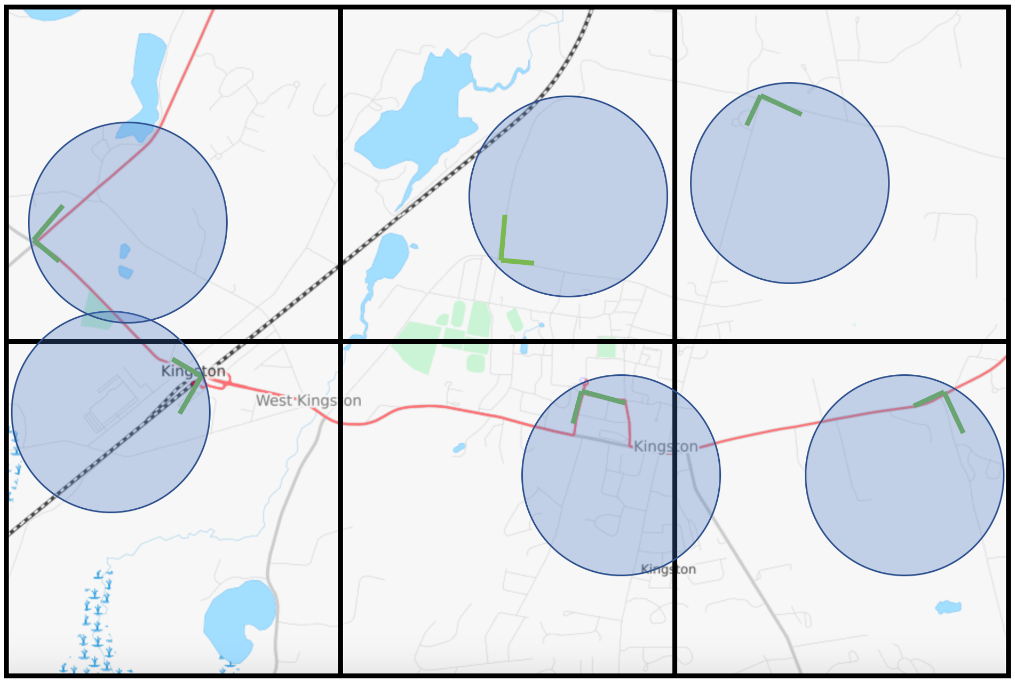


Figure 12. Work Distribution

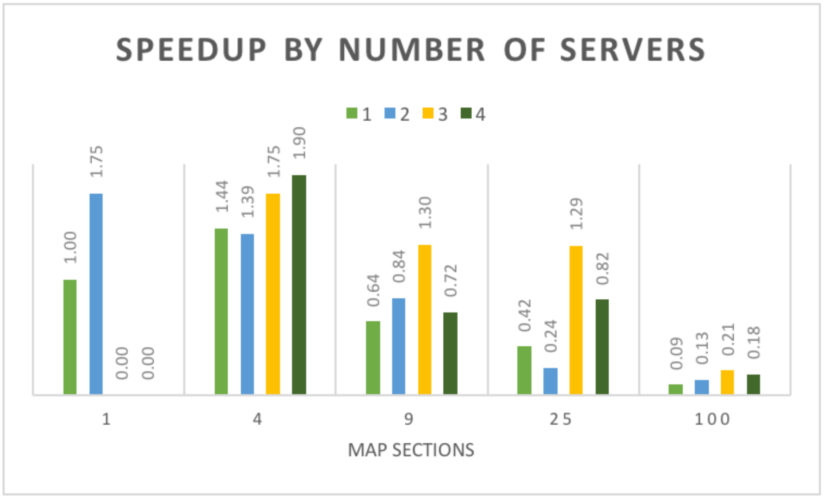


Figure 13. Distributed Speedup

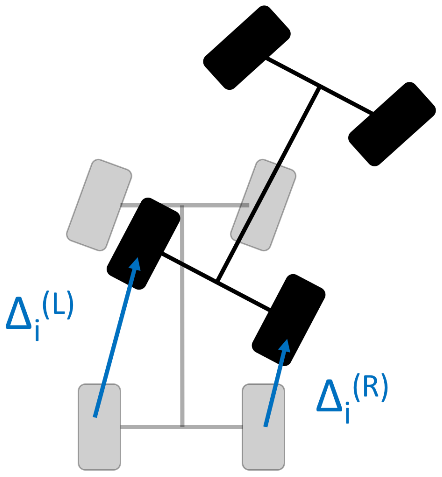


Figure 14. Differential Odometry by Rear Wheel Motion

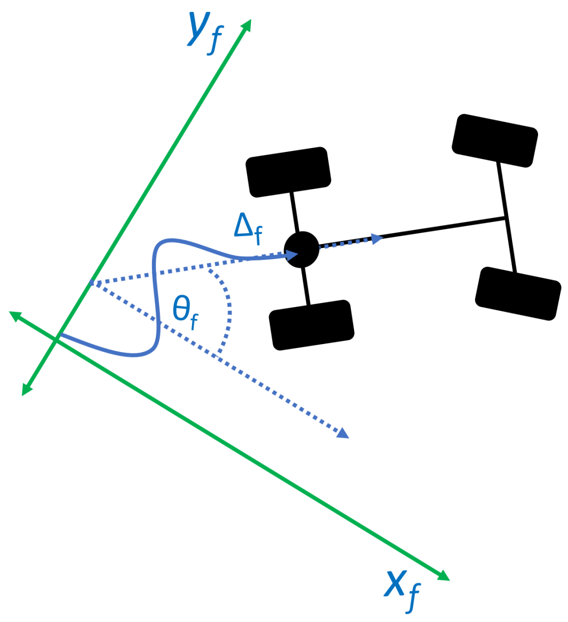
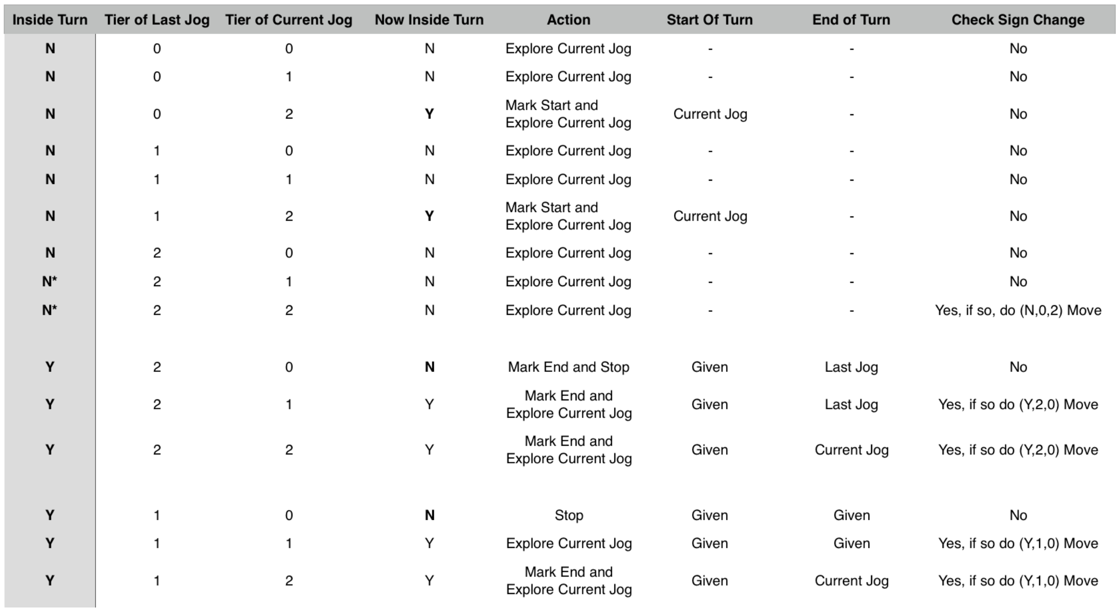


Figure 15. Change in distance and direction are given in a reference frame of Feature f_k.



Inside Turn	Tier of Last Jog	Tier of Current Jog	Now Inside Turn	Action	Start Of Turn	End of Turn	Check Sign Change
N	0	0	N	Explore Current Jog	-	-	No
N	0	1	N	Explore Current Jog	-	-	No
N	0	2	**Y**	Mark Start and Explore Current Jog	Current Jog	-	No
N	1	0	N	Explore Current Jog	-	-	No
N	1	1	N	Explore Current Jog	-	-	No
N	1	2	**Y**	Mark Start and Explore Current Jog	Current Jog	-	No
N	2	0	N	Explore Current Jog	-	-	No
N*	2	1	N	Explore Current Jog	-	-	No
N*	2	2	N	Explore Current Jog	-	-	Yes, if so, do (N,0,2) Move
Y	2	0	**N**	Mark End and Stop	Given	Last Jog	No
Y	2	1	Y	Mark End and Explore Current Jog	Given	Last Jog	Yes, if so do (Y,2,0) Move
Y	2	2	Y	Mark End and Explore Current Jog	Given	Current Jog	Yes, if so do (Y,2,0) Move
Y	1	0	**N**	Stop	Given	Given	No
Y	1	1	Y	Explore Current Jog	Given	Given	Yes, if so do (Y,1,0) Move
Y	1	2	Y	Mark End and Explore Current Jog	Given	Current Jog	Yes, if so do (Y,1,0) Move

Figure 16. Turn State Transition Table

CHAPTER 3

Testing Proposal

Below we discuss an experiment design for the RoadMapLocate system. The experiment will be tested off-line after recording GPS coordinates and logging ABS Wheel Speeds and timestamps from the CAN bus of a consumer vehicle. In order to carry out the RoadMapTrack and RoadMapFilter phases of the overall RoadMapLocate system, experimental data must be generated from rear wheel odometry over course where OSM data is available. The Arduino-bound RoadMapTrack system will record this data for analysis offline. For a real-world control, we will use GPS and network location estimates taken from the Android operating system interface. After processing the odometry data with RoadMapFilter, we will analyze the results along the Performance Criteria stated below.

3.1 Performance Criteria

The most common performance measure is the physical distance between position estimates and a "ground-truth" position, which is usually measured by a GPS device or corrected GPS track. We certainly care about that measurement here, but there are several additional measures we will consider as listed below.

Extent of Map In the literature, the test map is usually only large enough to contain the entire test track, and occasionally the size of the state space itself may not even be cited as a relevant performance criterion. However, the extent of the search on which a successful map-based localization strategy is performed is possibly the second most important measure of performance. Bigger map is better performance.

Computation Time We will benchmark each of the RoadMapLocate subcomponents on their respective hardware. RoadMapHash, executed on on an Intel I7 (6th Gen) with 32GB RAM time, should be evaluated for runtime as a function of N nodes in the roadmap. RoadMapTrack, executed on an ATmega328 with 2KB RAM, should be evaluated mostly for the speed of I/O operations. We are specifically interested in Bluetooth transfer rate. Finally, we expect RoadMapFilter to be run on a 1.6GHz quad-core Qualcomm Snapdragon 821 processor with 4GB of RAM, and we will estimate each particle update period in seconds.

Initial Position and Latency We want to quantify the necessary number of features recognized in order to accurately determine a position. This will be a function of map extent as well as the distribution of initial hypothesis across that extent.

Data Size This is related to the extent of map necessary for localization, but in this case, we seek to determine the geographical extent that could be stored on a consumer SD card so that the mobile device need not have Internet connectivity. Current estimates show that for an extent of approximately 100 square kilometers (Aquidneck Island) the required storage is 74KB. Assuming the rest of the 9 million square kilometers in the United States have similar road density, this translates to an estimated 6.5GB of storage space for the entire U.S.[1]

[1]We note that this estimate is pessimistic and the western United States encompasses a geographical extent with less road density.

CHAPTER 4

Conclusions and Future Research

Most map data-based localization strategies only tacitly recognize curvature as a road's fundamental feature. This research, in addition to developing recognizable turn features in the RoadMapHash algorithm, also contributes a unique technique to characterizing an entire roadmap based on an instinctual turn-by-turn approach to navigation. With modern sensors, small form factor computers, and a distributed computation environment, the success of the fully implemented RoadMapLocate technique is realistic in the near future.

Future research may pertain directly to the performance criteria listed in the Testing Proposal. However, we make acknowledge several additional items we would like to consider.

Data

The road features extracted from curvature can be augmented including the change in altitude between turns. This could reduce the load factor in the hashtables. Along that vein, road slope can be included when calculating feature keys so that distances covering any elevation during a real world drive will be more accurately represented than in the current flat Earth model. Additional road peculiarities, such as road surface width or speed limits might be of value.

Algorithmic Analysis

The majority of the work here concentrated on RoadMapHash, but the other components of the RoadMapLocate deserve significant analysis also. Because the error of each wheel speed is modeled as zero-mean Gaussian, we must derive the propagated error in the ultimate motion model, which includes sine and cosine functions, as a specific exponential probability distribution. Additionally, the scaled rounding technique used in RoadMapHash to determine how to round lookup keys must be properly parameterized, and adequately reflect error distributions in distance and heading changes.

Application Environments

The current operating environment for the RoadMapLocate system is a consumer vehicle with rear wheel speed sensors. However, we recognize several applications that could benefit from this research. For instance, visual odometry, performed either from a land-based or aerial platform, could identify road segments and benefit from the virtual measurements provided by RoadMapHash as well. Indoor navigation offers similar constraints as road-bound navigation and may benefit from a similar strategy. Finally, navigation in mines and tunnels, where GPS signals cannot travel but which may be entirely charted, could benefit from this strategy.[1]

List of References

[1] P. Debanne, J.-Y. Hervé, and P. Cohen, “Global Self-Localization of a Robot in Underground Mines,” in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5, Orlando, FL, USA, Oct. 1997, pp. 4400–4405 vol.5.

BIBLIOGRAPHY

"Progressive Taxi Instructions," 2017. [Online]. Available: https://www.skybrary.aero/index.php/Progressive_Taxi_Instructions

Advanced Navigation, "OBDII Odometer Reference Manual," Web, 2014. [Online]. Available: http://www.advancednavigation.com.au/sites/advancednavigation.com.au/files/obdii_odometer_reference_manual.pdf

Arkin, E. M., Chew, L. P., Huttenlocher, D. P., Kedem, K., and Mitchell, J. S. B., "An Efficiently Computable Metric for Comparing Polygonal Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 209–216, Mar 1991.

Badino, H., Huber, D., and Kanade, T., "Real-Time Topometric Localization," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA, May 2012, pp. 1635–1642.

Ballard, D. H., "Readings in computer vision: Issues, problems, principles, and paradigms," Fischler, M. A. and Firschein, O., Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, ch. Generalizing the Hough Transform to Detect Arbitrary Shapes, pp. 714–725. [Online]. Available: http://dl.acm.org/citation.cfm?id=33517.33574

Bhosle, U., Chaudhuri, S., and Dutta Roy, S., "A Fast Method for Image Mosaicing Using Geometric Hashing," *IETE Journal of Research*, vol. 48, no. 3-4, pp. 317–324, 2002.

Borenstein, J., Everett, H., and Feng, L., "Where am I? Sensors and Methods for Mobile Robot Positioning, Tech. Rep. 120, 1996.

Bruce, J., Wawerla, J., and Vaughan, R., "The SFU mountain dataset: Semi-structured woodland trails under changing environmental conditions," in *Proceedings of IEEE International Conference on Robotics and Automation 2015, Workshop on Visual Place Recognition in Changing Environments*, Seattle, WA, USA, 2015.

Carboni, D., Manchinu, A., Marotto, V., Piras, A., and Serra, A., "Infrastructure-Free Indoor Navigation: A Case Study," *Journal of Location Based Services*, vol. 9, no. 1, pp. 33–54, 2015.

Chaudhuri, S., Chatterjee, S., Katz, N., Nelson, M., and Goldbaum, M., "Detection of Blood Vessels in Retinal Images Using Two-Dimensional Matched Filters," *IEEE Transactions on Medical Imaging*, vol. 8, no. 3, pp. 263–269, Sep 1989.

Chen, H. and Bhanu, B., "Efficient Recognition of Highly Similar 3D Objects in Range Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 1, pp. 172–179, 2009.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: The MIT Press, 2009.

Debanne, P., Hervé, J.-Y., and Cohen, P., "Global Self-Localization of a Robot in Underground Mines," in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5, Orlando, FL, USA, Oct. 1997, pp. 4400–4405 vol.5.

Dhiman, N. K., Deodhare, D., and Khemani, D., "A review of path planning and mapping technologies for autonomous mobile robot systems," in *Proceedings of the 5th ACM COMPUTE Conference: Intelligent Scalable System Technologies*, ser. COMPUTE '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:8. [Online]. Available: http://doi.acm.org/10.1145/2459118.2459121

Durrant-Whyte, H. F., "Uncertain Geometry in Robotics," *IEEE Journal on Robotics and Automation*, vol. 4, no. 1, pp. 23–31, Feb 1988.

Engelmann, V., "www.Open-GraphTheory.org," 2017. [Online]. Available: http://www.open-graphtheory.org/gxlformat.html

Extract.bbbike.org, "Planet.osm Extracts — BBBike.org," 2016, Accessed: 2016-2-21. [Online]. Available: http://extract.bbbike.org/

Facebook, "GraphQL," 2016. [Online]. Available: http://facebook.github.io/graphql/October2016/

Floros, G., van der Zander, B., and Leibe, B., "OpenStreetSLAM: Global Vehicle Localization Using OpenStreetMaps," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2013, pp. 1054–1059.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R., "Vision Meets Robotics: The KITTI Dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

Google, "Protocol Buffers — Google Developers," 2017. [Online]. Available: https://developers.google.com/protocol-buffers/

Grabler, F., Agrawala, M., Sumner, R. W., and Pauly, M., "Automatic Generation of Tourist Maps," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 100:1–100:11, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1360612.1360699

Greitans, M., Pudzs, M., and Fuksis, R., "Object Analysis in Images Using Complex 2D Matched Filters," in *IEEE EUROCON 2009*, May 2009, pp. 1392–1397.

Grossmann, E. and Santos-Victor, J., "Uncertainty Analysis of 3D Reconstruction from Uncalibrated Views," *Image and Vision Computing*, vol. 18, no. 9, pp. 685–696, 2000.

Groves, P. D., *Principles of GNSS, Inertial, and Multi-Sensor Integrated Navigation Systems Second Edition.* Boston: Artech House, 2013.

Hagerup, T., "Planar Depth-First Search in O(log n) Parallel Time," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 678–704, 1990. [Online]. Available: https://doi.org/10.1137/0219047

Harvey, C., "New Algorithms for Automated Astrometry," Ph.D. dissertation, University of Toronto, Toronto, Ont., Canada, 2004.

Himsolt, M., *GML: A Portable Graph File Format.* [Online]. Available: https://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf

Hoffman, C., "China's Space Threat: How Missiles Could Target U.S. Satellites," 2009, Accessed: 2016-2-21. [Online]. Available: http://www.popularmechanics.com/space/satellites/a1782/4218443/

Jensfelt, P. and Kristensen, S., "Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, Oct. 2001.

Jiang, X., Broelemann, K., Wachenfeld, S., and Kruger, A., "Graph-Based Markerless Registration of City Maps Using Geometric Hashing," *Computer Vision and Image Understanding*, vol. 115, no. 7, pp. 1032–1043, July 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2010.12.014

Jungnickel, D., *Graphs, Networks and Algorithms*, 4th ed. New York, NY, USA: Springer, 2014.

Kalicinski, M., "RapidXml," 2009. [Online]. Available: http://rapidxml.sourceforge.net/

Kalman, R. E., “A New Approach to Linear Filtering and Prediction Problems,” *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

Karlsson, R. and Gustafsson, F., “The Future of Automotive Localization Algorithms: Available, Reliable, and Scalable Localization: Anywhere and Anytime,” *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 60–69, March 2017.

Karney, C. F. F., “GeographicLib,” 2015. [Online]. Available: https://geographiclib.sourceforge.io/

Kiriy, E. and Buehler, M., “Three-State Extended Kalman Filter for Mobile Robot Localization,” Montreal, Que., Canada, Tech. Rep., 2002.

Kleitman, D. “Planarity and Coloring.” 2004. [Online]. Available: http://www-math.mit.edu/~djk/18.310/18.310F04/planarity_coloring.html

Konheim, A. G., *Hashing and the Secure Distribution of Digital Media*. Hoboken, NJ, USA: John Wiley and Sons, Inc., 2010, pp. 320–323. [Online]. Available: http://dx.doi.org/10.1002/9780470630617.ch19

Kuipers, B., “Multiple Ontologies for Spatial Exploration and Mapping,” in *Proceedings of the Second ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, ser. ISA ’10. New York, NY, USA: ACM, 2010, pp. 24–24. [Online]. Available: http://doi.acm.org/10.1145/1865885.1865892

Kuipers, B. and Byun, Y.-T., “A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations,” *Robotics and Autonomous Systems*, vol. 8, no. 1, pp. 47 – 63, 1991.

Kuipers, B. J., “Representing Knowledge of Large-Scale Space,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977.

Kuipers, B. J. and Byun, Y., “A Qualitative Approach to Robot Exploration and Map-Learning,” in *Workshop on Spatial Reasoning and Multi-Sensor Fusion*, St. Charles, IL, USA, Oct. 1987, pp. 390–404.

Leibowitz, N., Fligelman, Z. Y., Nussinov, R., and Wolfson, H. J., “Multiple Structural Alignment and Core Detection by Geometric Hashing,” in *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, Heidelberg, Germany, 1999, pp. 169–177.

Li, W. and Leung, H., “Constrained Unscented Kalman Filter Based Fusion of GPS/INS/Digital Map for Vehicle Localization,” in *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, vol. 2, Shanghai, China, Oct. 2003, pp. 1362–1367.

Lifshits, M., Rivlin, E., and Rudzsky, M., "Image-Based Navigation on a Chip," in *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference*, vol. 1, Como, Italy, May 2004, pp. 504–509 Vol.1.

Lin, S. S., Lin, C. H., Hu, Y. J., and Lee, T. Y., "Drawing Road Networks with Mental Maps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 9, pp. 1241–1252, Sept 2014.

Liu, J. S. and Chen, R., "Sequential Monte Carlo Methods for Dynamic Systems," *Journal of the American Statistical Association*, vol. 93, pp. 1032–1044, 1998.

Liu, M. Y., Tuzel, O., Veeraraghavan, A., and Chellappa, R., "Fast Directional Chamfer Matching," in *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA, June 2010, pp. 1696–1703.

Lundgren, M., Stenborg, E., Svensson, L., and Hammarstrand, L., "Vehicle Self-Localization Using Off-the-Shelf Sensors and a Detailed Map," in *Proceedings of the 2014 IEEE Intelligent Vehicles Symposium*, Ypsilanti, Michigan, USA, June 2014, pp. 522–528.

Members, O., "OGC® Standards and Supporting Documents," 2017. [Online]. Available: http://www.opengeospatial.org/standards

Merriaux, P., Dupuis, Y., Vasseur, P., and Savatier, X., "Wheel Odometry-Based Car Localization and Tracking on Vectorial Map," in *Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, Oct. 2014, pp. 1890–1891.

Mohammed, S. and Bernard, M., *Graph File Formats*, 2004. [Online]. Available: http://www2.sta.uwi.edu/~mbernard/research_files/fileformats.pdf

Moreno Maza, M., "LL(1) Grammars," 2004. [Online]. Available: http://www.csd.uwo.ca/~moreno/CS447/Lectures/Syntax.html/node14.html

Mugan, J. and Kuipers, B., "Autonomous Learning of High-Level States and Actions in Continuous Environments," *IEEE Transactions on Autonomous Mental Development*, vol. 4, no. 1, pp. 70–86, March 2012.

Najjar, M. E. B. E. and Bonnifait, P., "Road Selection Using Multicriteria Fusion for the Road-Matching Problem," *Proceedings of the IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 279–291, June 2007.

National Coordination Office for Space-Based Positioning Navigation and Timing, "Gps.gov: Frequently Asked Questions," 2016, Accessed: 2016-2-21. [Online]. Available: http://www.gps.gov/support/faq/#off

Neo4j Staff, "The Database Model Showdown: An RDBMS vs. Graph Comparison - Neo4j Graph Database," 2017. [Online]. Available: https://neo4j.com/blog/database-model-comparison/

NXP Semiconductors, "GPS, LNA, Sensitivity, Jamming, Cohabitation, TTFF," Eindhoven, 2009, Accessed: 2016-2-21. [Online]. Available: http://www.nxp.com/documents/other/75016740.pdf

Petit, L., "MongoDB WiredTiger: Why We Switched Back to MMap," 2015. [Online]. Available: https://tech.c-radar.com/2015/10/12/mongodb-wiredtiger-why-we-switched-back-to-mmap/

ProofWiki Contributors, "Set Difference of Cartesian Products," Aug 2008. [Online]. Available: https://proofwiki.org/wiki/Set_Difference_of_Cartesian_Products

Ranganathan, P., Hayet, J.-B., Devy, M., Hutchinson, S., and Lerasle, F., "Topological Navigation and Qualitative Localization for Indoor Environment Using Multi-Sensory Perception," *Robotics and Autonomous Systems*, vol. 41, no. 2-3, pp. 137–144, 2002.

Raymond, E., "The Lost Art of C Structure Packing," 2017. [Online]. Available: http://www.catb.org/esr/structure-packing/

Rigoutsos, I. and Wolfson, H. J., "Geometric Hashing," *IEEE Computational Science and Engineering*, vol. 4, no. 4, pp. 9–9, Oct. 1997.

Santana, A. M., Sousa, A. A., Britto, R. S., Alsina, P. J., and Dantas de Medeiros, A. A., "Localization of a Mobile Robot Based on Odometry and Natural Landmarks Using Extended Kalman Filter," in *Proceedings of the 2008 International Conference on Informatics in Control, Automation and Robotics*, Funchal, Madeira, Portugal, 2008.

Schraffenberger, M. and Hervé, J. Y., "Agent Abilities in a Landmark-Based Mapping Model," in *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, Oct. 2006, pp. 2493–2498, Taipei, Taiwan.

Schutzberg, A., "Ten Things You Need to Know about Indoor Positioning," 2013, Accessed: 2016-2-21. [Online]. Available: http://www.directionsmag.com/entry/10-things-you-need-to-know-about-indoor-positioning/324602

Sim, K. S., "Rotation-Invariant Reference Point Location Detection Using Complex Filtering for Fingerprint Matching," *International Journal of Future Computer and Communication*, pp. 321–322, 2012.

StackExchange Contributors, “Since When is CAN Bus Mandatory for New Vehicles?” 2017. [Online]. Available: https://law.stackexchange.com/questions/1317/since-when-is-can-bus-mandatory-for-new-vehicles

Staff, N., “The Database Model Showdown: An RDBMS vs. Graph Comparison - Neo4j Graph Database,” 2017. [Online]. Available: https://neo4j.com/blog/database-model-comparison/

Strecha, C., Bronstein, A., Bronstein, M., and Fua, P., “LDAHash: Improved Matching with Smaller Descriptors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 66–78, Jan 2012.

Sutherland, K. T. and Thompson, W. B., “Localizing in Unstructured Environments: Dealing With the Errors,” *IEEE Transactions on Robotics and Automation*, vol. 10, no. 6, pp. 740–754, Dec 1994.

Tanase, M. and Veltkamp, R. C., “Part-Based Shape Retrieval,” in *Proceedings of the 13th Annual ACM International Conference on Multimedia*, Singapore, 2005.

Thrun, S., Burgard, W., and Fox, D., *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*, ser. Intelligent Robotics and Autonomous Agents. Cambridge, Massachusetts, USA: The MIT Press, Aug. 2005. [Online]. Available: http://www.worldcat.org/isbn/0262201623

Vanasse Hangen Brustlin Inc. et al, *Norwalk Transportation Management Plan: State Project DOT01020336PE*, 2014, ch. Intersection Design Template. [Online]. Available: http://www.ct.gov/dot/cwp/view.asp?A=3529&Q=542234

Wachenfeld, S., Broelemann, K., Jiang, X., and Kruger, A., “Graph-Based Registration of Partial Images of City Maps Using Geometric Hashing,” in *Proceedings of the 7th IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition*, ser. GbRPR ’09, Venice, Italy, 2009, pp. 92–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02124-4_10

Weiss, W. A. R., *An Introduction to Set Theory.* University of Toronto, 2008.

Weisstein, E. W., “Normal Difference Distribution,” MathWorld, A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/NormalDifferenceDistribution.html

Whaite, P. and Ferrie, F. P., “From Uncertainty to Visual Exploration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 10, pp. 1038–1049, Oct. 1991.

Whaite, P. and Ferrie, F. P., “Autonomous Exploration: Driven by Uncertainty,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 3, pp. 193–205, Mar 1997.

Wikipedia Contributors, “Data-centric programming language,” 2017. [Online]. Available: https://en.wikipedia.org/wiki/Data-centric_programming_language

Wikipedia Contributors, “Graph modelling language,” 2017. [Online]. Available: https://en.wikipedia.org/wiki/GraphModellingLanguage

Wikipedia Contributors, “Markov Property,” 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Markov_property&oldid=789272803,

Wikipedia Contributors, “OBD-II PIDs,” 2017. [Online]. Available: https://en.wikipedia.org/wiki/OBD-II_PIDs

Wikipedia Contributors, “Rounding,” 2017.

Wolfson, H. and Rigoutsos, I., “Geometric Hashing: An Overview,” *IEEE Computational Science and Engineering*, vol. 4, no. 4, pp. 10–21, 1997.

Yanchyshyn, R. “How-to Guide for OBDII Reader App Development.” May 2014. [Online]. Available: http://blog.lemberg.co.uk/how-guide-obdii-reader-app-development

Ying, F., Mooney, P., Corcoran, P., and Winstanley, A. C., “Using Shape Complexity to Guide Simplification of Geospatial Data for Use in Location-Based Services,” in *7th International Symposium on Location Based Services & TeleCartography*, Guangzhou, China, 2010, pp. 1–16. [Online]. Available: http://eprints.maynoothuniversity.ie/4985/

Yip, M. and Tencent Corporation, “Rapidjson,” 2015. [Online]. Available: http://rapidjson.org/index.html