Utility-based Map Reduction for Ground and Flight Vehicle Navigation

by

Theodore J. Steiner III

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2015

Author ..... Signature redacted
Department of Aeronautics and Astronautics
May 21, 2015

Certified by ..... Signature redacted
Jeffrey A. Hoffman
Professor of the Practice of Aerospace Engineering
Thesis Supervisor

Certified by ..... Signature redacted
John J. Leonard
Samuel C. Collins Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Certified by ..... Signature redacted
Dr. Paul A. DeBitetto
Principal Member of the Technical Staff, Draper Laboratory

Certified by .. Signature redacted
Dr. Babak E. Cohanim
Chief Scientist, MORSE Corp

Accepted by Signature redacted
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Utility-based Map Reduction for Ground and Flight Vehicle Navigation

by

Theodore J. Steiner III

Submitted to the Department of Aeronautics and Astronautics
on May 21, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Maps used for navigation often include a database of location descriptions for place-recognition (to enable localization or loop-closing), which permits bounded-error navigation performance. A standard localization system must describe the entire operational environment in its place-recognition database. A standard pose-graph-based simultaneous localization and mapping (SLAM) system adds a new place-recognition database entry for every new vehicle pose, which grows linearly and unbounded in time and thus becomes unsustainable.

To address these issues, this thesis proposes a new map-reduction approach that pre-constructs a fixed-size place-recognition database amenable to the limited storage and processing resources of the vehicle by exploiting the high-level structure of the environment and vehicle motion. In particular, the thesis introduces the concept of *location utility* – which encapsulates the visitation probability of a location and its spatial distribution relative to nearby locations in the database – as a measure of the value of potential localization or loop-closure events to occur at that location. While finding the optimal reduced location database is NP-hard, an efficient greedy algorithm is developed to sort all the locations in a map based on their relative utility *without* access to sensor measurements or the vehicle trajectory. This enables pre-determination of a generic, limited-size place-recognition database containing the N best locations in the environment.

A street-map simulator using city-map data and a terrain relative navigation simulator using terrestrial rocket flight data are used to validate the approach and show that an accurate map and trajectory reconstruction (pose-graph) can be attained even when using a place-recognition database with only 1% of the entries of the corresponding full database.

Thesis Supervisor: Jeffrey A. Hoffman
Title: Professor of the Practice of Aerospace Engineering

Thesis Supervisor: John J. Leonard
Title: Samuel C. Collins Professor of Mechanical and Ocean Engineering

Acknowledgments

I feel very privileged to have received such wonderful support and guidance from so many great advisors and mentors throughout the course of my graduate education at MIT and Draper Laboratory. Despite my projects changing several times throughout my graduate career, my advisor, Professor Hoffman, and his support has been the one constant factor in my academic life. His advice and guidance helped me navigate and overcome several project cancellations that threatened my research progress. I was also fortunate to have Professor Leonard offer to serve as a co-advisor partway into my doctoral studies, which gave my research additional depth and stability. It was truly an honor to be co-advised by such highly-regarded professors. I was also very lucky to be a member of two of the most distinguished research groups worldwide in their respective fields while at MIT, the AeroAstro Space Systems Lab and CSAIL Marine Robotics Group. The numerous colleagues I have worked with in these two groups are too many to name, but I'd especially like to thank Padraig Corcoran, Paul Huang, Liam Paull, and Matt Graham for their help with my thesis-related publications.

I had the honor of receiving a Draper Laboratory Fellowship for all five years of my graduate studies and the mentorship associated with it. My supervisor at Draper, Scott Rasmussen, repeatedly went above and beyond by helping me with my research, teaching me the value of technical foresight during the development process, and providing me with new opportunities, even occasionally infiltrating various projects at Draper to help me get my foot in the door. I also benefitted immensely from the mentorship of Tye Brady, Bobby Cohanim, Paul DeBitetto, Pete Lommel, Rich Madison, Steve Paschall, Brett Streetman, and many others.

I would also like to thank the many friends I made at MIT for keeping life fun and being there whenever I needed it, especially Farah, Connie, Andrew, and Dustin. And most importantly, I would like to thank my loving girlfriend, brother, and parents for their support and encouragement throughout this difficult but rewarding journey.

Assignment

In consideration of the research opportunity and permission to prepare this thesis by and at the Charles Stark Draper Laboratory, Inc., the author hereby assigns copyright of the thesis to The Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts and the Massachusetts Institute of Technology. The author hereby grants to MIT and Draper Laboratory permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

In mobile robotics, it is almost always important to know where a robot is located and pointing to perform even very simple tasks. Determination of the position and orientation of the vehicle, referred to as its *pose*, has long been an active area of research, and a variety of approaches have been proposed and successfully implemented.

At the simplest level, odometry systems provide a measurement of the relative motion of a vehicle. The most basic odometry sensor is a wheel encoder, which counts the revolutions of a wheel and provides an estimate of distance traveled over an interval of time. More complex odometry systems have been developed that utilize sensors such as cameras, accelerometers, and gyroscopes. These odometry systems can provide the full 6-degree of freedom relative motion (position and orientation in 3D space) for generic platforms, as they only require knowledge of the sensor position and orientation relative to the vehicle's coordinate frame, and do not require prior knowledge of the vehicle's parameters, such as its wheel diameter.

Because odometry measurements are computed relative to the previous vehicle state, these navigation systems suffer from error accumulation over time, called *drift*. This means that odometry alone will eventually grow too inaccurate for long-duration or persistent system navigation. To compensate for drift we must add a sensor that provides information on the vehicle pose in some external reference frame (absolute or

global pose), as opposed to the current pose relative to some previous pose (relative pose), which is provided by odometry. When available, position updates from a Global Positioning System (GPS) receiver provide convenient absolute measurements on Earth.

When GPS signals are unavailable or unreliable, such as inside buildings or on other planets, we need a different form of absolute position update. One way to do this is by computing the vehicle's pose relative to a map (called *localization*). If a map is unavailable ahead of time, we can construct one online using observations of our environment (called *mapping*). The map can either be defined in the global coordinate frame to directly provide absolute pose measurements or a local coordinate frame, providing absolute pose measurements accurate up to some map-tie error or gauge transform, which relates the map coordinate frame to the global coordinate frame. Navigation applications that do not require global positioning typically equate the map coordinate system with the global frame.

1.1 Navigation Databases

A navigational map can be thought of as a database associating locations in the world with descriptive data, which we will hereafter refer to as a *location database.* The location descriptor is tailored to the specific sensing modality used and must be sufficient to be matched to vehicle sensor data in a process called *data association.* In an ideal localization system, the vehicle sensor data is correlated against every location descriptor in the database, the database provides complete coverage of the environment, and all locations in the environment can be uniquely described. In the case of such an ideal system, the best estimate of the vehicle's position will be that corresponding to the strongest database match.

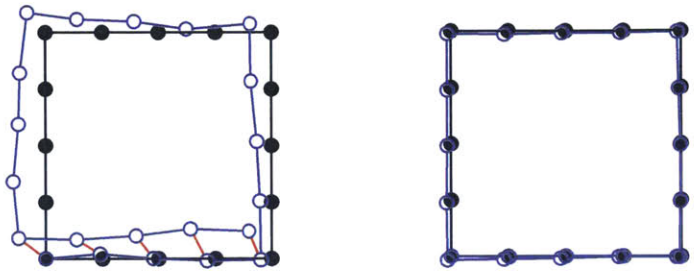
However, there are a number of challenges associated with the scalability of ideal localization systems. The most difficult challenge is the massive volume of data required to describe even a moderate sized environment, which becomes a burden both to store in memory and to search through during real-time operation. Additionally, as environments grow large, locations within them tend to become less distinct, making it difficult to find a distinctly correct match for the vehicle position.

1.1.1 Simultaneous Localization and Mapping

A vehicle can navigate using a reduced localization database as long as it has an odometry system capable of providing vehicle pose estimates even when the current vehicle position cannot be matched to the localization database. This situation is frequently encountered in the field of Simultaneous Localization and Mapping (SLAM), one of the central issues of mobile, autonomous robotics [1, 2]. In its most basic form, SLAM is the problem of a robot existing at an unknown location in an unknown world, where the robot simultaneously estimates both its surroundings and its pose relative to those surroundings.

Modern SLAM systems improve upon the results of pure odometry when trajectories return to previously visited locations through a process called *loop-closure*, which estimates and corrects for drift, as shown in Figure 1-1 [3]. A SLAM system builds its location database as it explores its environment, generally adding a new location entry for every estimated vehicle pose. To detect a loop-closure, the system must first identify that it has returned to a previously mapped location using the same data association techniques used for localization systems. When a loop-closure is detected, a constraint is added to the map encoding the relationship between the current and previous vehicle poses. The trajectory estimation can then be updated, including this additional constraint, typically by solving a nonlinear least-squares constraint

satisfaction optimization. For a physical analogy, one can consider the map to be a network of springs whose stiffnesses are proportional to the confidence of the odometry measurements. If the SLAM system adds a loop-closure spring that further strains the spring network, the network becomes stiffer (more certain).

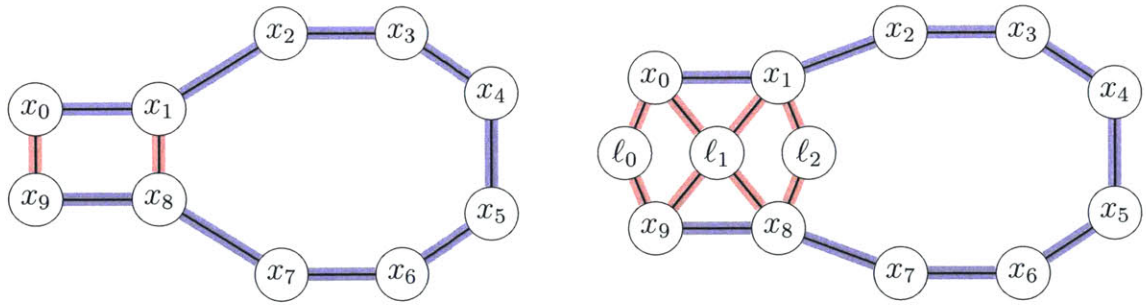


(a) Before loop-closure. (b) After loop-closure.

Figure 1-1: A vehicle travels around a square box counter-clockwise, beginning in the bottom left corner and ending in the bottom right corner, with the true trajectory shown in black and the estimated trajectory shown in blue. In (a), the vehicle's position estimate is the result of accumulating odometry measurements. However, the system detected five loop-closures (red) in the final leg of the trajectory. In (b), the optimizer has applied the loop-closure constraints, eliminating most of the drift from the earlier trajectory estimate.

1.1.2 Pose-Graph Representation

This work focuses on a popular SLAM representation called a "pose-graph," which represents the map and vehicle trajectory using a graph of connected poses, as shown in Figure 1-2a. Individual poses form the vertices of the graph, and constraints between poses form the edges. Consecutive poses are linked using odometry constraints, and location revisits are linked using loop-closure constraints between associated poses. All vertices and edges in the graph have some associated uncertainty and the goal is to reduce that uncertainty by adding sufficient constraints. The standard pose-graph representation associates each pose with an entry in the location database. An alternative formulation separates the vehicle poses and data association, instead associating location database entries with "landmarks," which are modeled as separate graph indices from the poses, as shown in Figure 1-2b.



(a) Standard Pose Graph

(b) Pose Graph with Landmarks

Figure 1-2: Simple pose-graphs, including loop-closures (left) or landmark measurements (right). Poses, x_i, and landmarks, ℓ_j, are represented as circles, odometry constraints are blue, and loop-closure and landmark constraints are red.

In addition to being a valuable representation for SLAM problems, pose-graphs can be used to incorporate odometry measurements into localization systems, relaxing the requirement that the location database must describe every possible vehicle location in the environment. Using odometry measurements also improves robustness to failed data association, where the vehicle is unable to localize itself accurately. The pose-graph representation is described in greater detail in Chapter 2.

1.1.3 Data Volume Challenges

One major challenge with pose-graph systems is the large data storage required for the associated location database. In a standard localization-only system, the database size is directly proportional to the number of locations in the environment at which the vehicle could possibly be located, and thus is bounded for a fixed-size environment. In order to enable data association for loop-closure detection in SLAM, observational data is typically retained for every past pose or landmark. While compression schemes can be used to greatly reduce the data storage required to describe a location, the total storage required still grows linearly with the size of the environment for localization or the total number of poses (or landmarks) for SLAM.

Large navigation databases can additionally cause challenges for systems without limiting memory constraints. For example, a larger location database may take longer to search within than a small database, potentially increasing processing requirements. Location descriptions in an especially large navigation database may also become overly similar, making them difficult to uniquely match to sensor data. Furthermore, large-sized navigation databases are challenging to transmit over limited-bandwidth networks, such as when sending a vehicle a map of its environment over a wireless connection.

Recent work has shown that the quality of the constraints is more important than the quantity, and that adding too many constraints may harm solver performance [4], making constraint selection an important consideration in modern pose-graph localization and SLAM systems. Constraint quality here refers to the *information content* of the measurement, or roughly its precision and proximity to other constraints in the pose-graph. This means that long segments of poses can be connected solely using odometry constraints, with constraints involving the location database occurring sparsely. However, in practice, such efficient pose-graph localization and SLAM systems maintain a complete location database despite actually using this database sparingly in order to maintain solver performance, and thus the location database is vastly underutilized. This thesis provides a means to predict where the most valuable localization or loop-closure events will occur in an environment prior to vehicle navigation. This enables vehicle navigation with an efficient, reduced-size location database containing only the most valuable locations in the vehicle's operational environment.

1.2 Exploiting Vehicle and Environment Structure

When people enter new and unfamiliar environments, they often seek out a map, such as from an information stand or a smart phone. Figure 1-3 shows a comparison of a simplified street map and dense map of Harvard Square in Cambridge, Massachusetts – a notoriously difficult area for first-time visitors to navigate – representing two options readily available to a mobile device user. Notice that the simplified map has already effectively selected the most valuable information and presented it in a concise form, while the dense map contains a great deal of additional, irrelevant data. Provided with only very limited information – the topology of the traversable paths available to them and a few basic landmarks in the map – a typical human can perform complex navigation tasks using only the simplified map of Figure 1-3a. People can perform these tasks sufficiently well that even when presented with the detailed satellite imagery in Figure 1-3b, many still prefer the simplified map representation, which may better correspond to their own "mental map" of the region [5]. We say that this simplified map is optimally constructed if any additional piece of data would be less valuable than any existing piece of data to the user for purposes of navigation within the mapped environment.

The following example of human navigation in an unfamiliar environment motivates the issues that must be addressed to implement autonomous navigation with reduced-size location databases. Consider the case of designing a printed map for tourists visiting a city, which will display both the topology of the streets in the city and images of local landmarks. Unfortunately, this city does not have readable street signs conveniently at every corner, so the tourist only has the map and the set of images available to help him navigate. Even if the tourist knows where he started walking in the map, over time he will grow less certain of where he is, and eventually will become lost without some other external input. However, every time he can match his current view to a picture on the map of a landmark, he knows where he is again. Placing too many pictures on the map means we have to make them too

(a) Street map (b) Dense map

Figure 1-3: Comparison of a simplified and dense map of Harvard Square in Cambridge, Massachusetts. Despite providing less information, the simplified map representation is still often preferred for navigation tasks.

small to be recognizable. Furthermore, if there are too many pictures, it will take the tourist too long to look through them all. Thus we determine that we can only afford to include ten pictures on our map. How should we select which ten pictures are the most valuable to *any* tourist, even without knowing their intended routes?

There are a few factors we will want to take into consideration when selecting these images. First, the images need to be unique and recognizable. If a tourist is able to match his locations to an image, it should place him uniquely on the map. This is generally a property of the specific sensing system used – in this case, the tourist's eyes. Second, the images should be of landmarks the tourist is likely to see. For example, a typical tourist in New York City is more likely to see Times Square than a random alleyway, even if that alleyway happens to have some unique graffiti painted along it. Essentially, we want our few selected images to be used by the tourist as frequently as possible, so we place them in high traffic areas. Third, our selected images should be well-distributed in the environment rather than tightly clustered. While our tourist might be likely to walk through Times Square, having a second image of a different view of or very near to that same location will provide him with

very little added value. Instead, we want to select images corresponding to locations where the tourist is expected to be the most lost (i.e., maximally distant from his last landmark sighting, assuming he grows increasingly uncertain of his location with every additional step), so we should place each image as far away from the other images as possible. In short, to assemble the most useful set of ten images for our map, the landmarks in the images should be recognizable, in heavily trafficked areas, and evenly distributed about the environment.

The problem of constructing a reduced-size map for autonomous vehicle navigation is analogous to the optimal tourist map problem. A more relevant example is the development of a global "Google Maps-like" service [6], from which a vehicle can query a routing network and location-recognition database for a specified region. Due to network bandwidth constraints, we want to transmit only the most valuable locations to the vehicle, but we do not know the vehicle's intended route due to privacy or security concerns. Another example is the preparation of a terrain database for precision landing on Mars. Storage and processing time constraints dictate that we can pre-load only the N most valuable terrain landmarks onto the landing vehicle, but the true vehicle trajectory is unknown due to aerodynamic disturbances and modeling errors. In both of these cases, a location database must be computed *prior* to vehicle navigation in order to facilitate localization or loop-closure measurements when navigating an unknown route within a region.

This thesis introduces the notion of *location utility*, which is a measure of a location's informativeness based on its potential localization or loop-closure measurements, and is defined independently of any specific vehicle trajectory. The location utility encapsulates the structure of the environment and the vehicle motion provided by the knowledge of the vehicle's path planner and a model of potential paths in the environment. The location utility is then used to select a limited-size database of location

descriptors from all available locations in the environment that maximizes the total utility of the database for vehicle navigation. The contributions of this thesis show how limited, but commonly available, prior information can be used to significantly reduce the amount of data required for vehicle navigation.

1.3 Research Objectives

This thesis explores the case of navigating a vehicle moving in a known or partially known environment under resource constraints that limit the number of locations that can be stored in the vehicle's memory and correlated to sensor data. The core problem is to determine which locations in the environment should be stored in the vehicle's onboard location database, such that the vehicle can most accurately navigate according to an optimal path planner rather than being required to divert along suboptimal paths.

This leads us to the following research questions:

1. What properties of a map location determine the information content of a localization or loop-closure measurement?
2. How can a database of location descriptors used for vehicle pose estimation be selected independently of the true vehicle trajectory?
3. How much map data is required to navigate within a region?

The overarching objective of this thesis is to define a metric of location value for vehicle navigation based upon properties of the operational environment rather than the specific vehicle trajectory, which is unobservable at the time of map database selection. The specific research objectives are:

1. Identify the characteristics that determine the value of mapped locations in an environment for navigation, independently of the specific vehicle trajectory. Using these characteristics, define a metric of location utility for navigation.

2. Formulate map location selection as an optimization problem to allow constrained systems to maximally utilize their limited onboard resources.

3. Demonstrate that a vehicle can feasibly navigate accurately even when using a greatly reduced location database.

1.4 Thesis Overview

The remainder of the thesis addresses these objectives, proceeding as follows. Chapter 2 provides additional background on simultaneous localization and mapping (SLAM) and related work on measurement selection and map reduction. Chapter 3 introduces a novel metric for trajectory-independent location utility, applicable to both pose- and landmark-based graphical formulations of SLAM. Chapter 4 uses this location utility metric to formulate an optimal map database selection problem. Because the resulting formulation is NP-hard and thus computationally intractable, an efficient greedy approximation algorithm is presented. Chapter 4 additionally provides a method of further reducing computation for especially large maps or updating a portion of an existing map by dividing the map into several "tiles."

The value of utility-based database selection is demonstrated using two applications. Chapter 5 describes a pose-graph simulator for vehicles driving in 2D street networks and demonstrates navigation using maximum-utility greedy location selection to pre-determine potential loop-closure locations. Chapter 6 applies the utility-based location selection to 3D landmark-based terrain relative navigation for spacecraft planetary landing, which is traditionally implemented on highly constrained hardware systems. Chapter 6 additionally describes a factor-graph-based incremental smoothing framework for terrain relative navigation, which is used to demonstrate the utility-based landmark selection approach. Chapter 7 closes the thesis with some conclusions and lists some open questions left for future work.

Chapter 2

Background & Literature Review

This chapter provides background relevant to the thesis and places the contributions in the context of related work. The contributions of this thesis are applicable to either localization or SLAM, the only difference is that in the case of SLAM, a location prior is not included with each entry in the localization database, making the problem more challenging. Chapter 5 presents results for the case of 2D SLAM, and Chapter 6 presents results for 3D localization, but both applications rely on a pose-graph representation, which originated in the field of SLAM, and thus the related work is presented primarily in this context.

This chapter will first provide some background and major developments from the field of SLAM, leading up to the now commonplace pose-graph representation of SLAM, which formulates the problem as an optimization over the trajectory (localization) or trajectory and map (SLAM). A standard pose-graph localization or SLAM formulation utilizes two classes of constraints: odometry constraints and loop-closure constraints. Section 2.2 provides some background on odometry systems. While the contributions of this thesis are relevant to any odometry system, they are enhanced by modern visual-inertial odometry algorithms, which allow a vehicle to navigate for long distances with low error growth. Section 2.3 provides background on appearance-

based loop-closure detection systems, which correlate vehicle position to a location database. While the contributions of the thesis are relevant to any loop-closure detection approach, we focus on vision-based approaches, which achieve high quality results but suffer from challenges associated with large location databases.

As discussed in the previous chapter, localization and SLAM suffer from challenges associated with large maps. Section 2.4 details a variety of approaches proposed in the literature for handling these challenges and places the thesis contributions in the context of these approaches. The main contribution of this thesis is a method of reducing the size of the map location database for a given environment prior to navigating by exploiting vehicle routing structure and environment structure. Routing structure is extracted from the environment independently of the true vehicle trajectory using knowledge of the vehicle's routing objective function (e.g., minimizing route time). Section 2.5 gives some general background on graph-based planning algorithms, applicable to environments with network structure, as well as some background on route planning in a pose-graph. While network environments encode a large amount of exploitable structure, a network environment is not a requirement of the proposed location database reduction approach. Section 3.3 extends the reduction approach to more complex planning and guidance systems.

2.1 Simultaneous Localization and Mapping

As previously discussed at a high level in Section 1.1.1, SLAM is the problem in which a robot simultaneously estimates both a map of its surroundings and its pose relative to that map. Both metric and topological approaches have been taken to represent this map of the environment [7]. Metric map representations capture the geometric properties of the environment, typically representing the map in a globally consistent coordinate frame [8, 9, 10], and directly enable typical robotic planning tasks, such as route planning in metric space. Topological map representations instead treat the environment as a network of connected places [11, 12, 13]. These representations

are often biologically motivated, as humans [5, 14] and animals [15, 16] typically consider paths through the environment in terms of the intermediate locations they will need to pass through rather than in terms of metric distances. Topological maps thus focus more on the connectivity of the environment, as opposed to metric maps, which aim for metrically correct reconstructions, although the representations are often combined in various forms [17, 18, 19, 20]. The utility-based location selection approach proposed by this thesis for location database reduction is applicable to both metric and topological map representations. The remainder of this section will briefly give some of the basics of SLAM and the highlights of vision-based SLAM research over the past 30 years. An excellent general introduction to SLAM can be found in either [1] or [2] and [21].

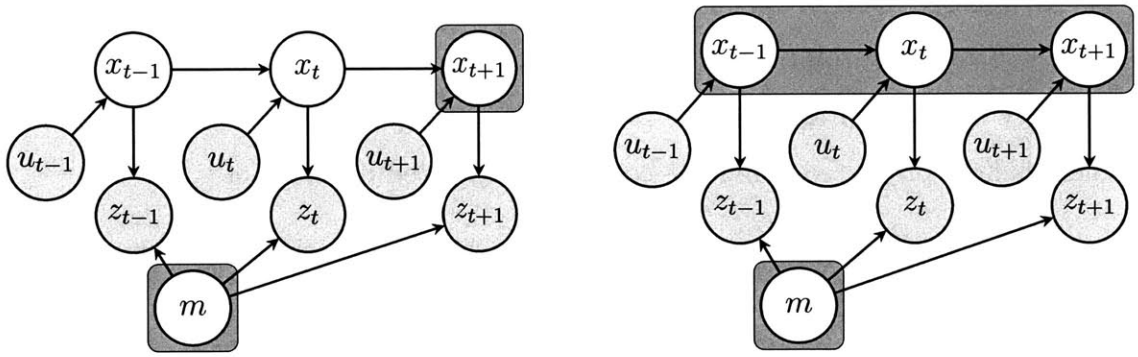
2.1.1 Introduction to SLAM

In the localization and SLAM problem, robot motions and sensor observations are imperfect, containing some model or measurement uncertainty. Due to this inherent uncertainty of its inputs, SLAM is best treated in a probabilistic framework. The problem is here presented using the commonly accepted notation of [1].

First, we define our vehicle state, here being position and orientation (pose), as x. Denoting time by t and the terminal time by T, we can describe the sequence of poses as $\boldsymbol{x}_{1:T} = \{x_0, x_1, \ldots x_T\}$. An odometry measurement provides the change from one state to the next, $u_t = x_t \ominus x_{t-1} + \varepsilon_{odom}$, with some uncertainty $\varepsilon_{odom} \sim \mathcal{N}(0, \Sigma_{odom})$, where operator $\ominus$ indicates that measurement u_t is in the frame of pose x_{t-1}. We can describe the sequence of odometry measurements as $\boldsymbol{u}_{1:T} = \{u_1, u_2, \ldots, u_T\}$.

In a pose-graph formulation of SLAM, the map, m, consists of the previous poses in the trajectory, $\boldsymbol{x}_{1:t-1}$. In a landmark formulation of SLAM, which is especially common in visual SLAM, the map, m, consists of discrete *landmarks*, which are typically point locations in space. At each state, x_t, the robot makes a set of measurements, $\boldsymbol{z}_t$, of the relative locations of poses or landmarks in m. We can describe the sequence of sensor measurements as $\boldsymbol{z}_{1:T} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_T\}$.

The goal of SLAM is to estimate the posterior distribution of the vehicle locations, $\boldsymbol{x}$, and the map, m, given measurements, $\boldsymbol{z}$, and odometry, $\boldsymbol{u}$. The two main paradigms of SLAM are the "online SLAM problem" and the "full SLAM problem" [1]. The online problem solves for the posterior over only the current pose: $P(x_t, m|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t})$, as shown in Figure 2-1a. In the full problem, we solve for the posterior over the entire path: $P(\boldsymbol{x}_{1:t}, m|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t})$, as shown in Figure 2-1b. The online problem is generally implemented using an estimation filter, while the full problem is implemented using a smoother.



(a) Online SLAM.

(b) Full SLAM.

Figure 2-1: Bayesian graphical models of online and full SLAM [1].

2.1.2 Online SLAM

The origin of SLAM research dates back to the late 1980s, with introductory work by [8, 22], who created stochastic maps of spatial relationships between objects in an unknown world in the framework of an Extended Kalman Filter (EKF) [23]. This approach, later referred to as EKF-SLAM, incorporated uncertain landmarks into

the EKF state vector to be estimated with the vehicle state. While [8, 22] used EKF-SLAM in conjunction with beacons in uncertain locations, [24] used straight line segments detected by a scanning laser rangefinder as landmarks, and [9] used sonar sensors, which required more rigorous data association. During the 1990s and early 2000s, EKF-SLAM was the dominant approach to SLAM, perhaps peaking with the work of [10], which proved that the EKF-SLAM map converges monotonically to a relative map with zero uncertainty. They additionally showed improved results by initializing landmarks in a tentative list and promoting them into the EKF state vector only after a landmark receives sufficiently high quality measurements.

Up to this point, the majority of SLAM systems had operated in a 2D (3 degrees-of-freedom: lateral motion and heading) world using range-and-bearing measurements (such as sonar or laser rangefinders). [25] extended this work into the realm of vision sensors by using a stereo camera, which provide ranging information using triangulation between observations from two cameras with a known separation distance. [26, 27] later gave the first bearing-only EKF-SLAM system using a single, monocular camera in 3D space, which they called MonoSLAM.

EKF-SLAM suffers from three fundamental and significant limitations. First, all probability distributions in an EKF are assumed to be Gaussian, and thus all measurements must be linearized. This limitation is made worse by the fact that the EKF estimates only the current vehicle and map state, meaning measurements are all linearized about the current best estimate, which is uncertain and therefore inaccurate. Second, the EKF is inherently sensitive to failures of data association, because any errors are marginalized into the state estimate on the next iteration (similarly to measurement linearization errors), meaning they can no longer be corrected for. Both of these issues can cause the solver to grow inconsistent, meaning its error covariance estimate does not accurately describe the true state error (i.e., the filter grows over-confident), which can cause the filter to diverge.

The third limitation is that the computational complexity of an EKF grows quadratically with the length of the state vector, making the system intractable for large environments. This issue of computational complexity growth has been partially addressed by techniques such as using submaps, which place only the most relevant portion of the map in the state vector at any given time [28, 29]. These systems use several decorrelated submaps to bound the complexity growth of the EKF, which must invert the covariance matrix on every iteration. However, as the correlation between landmarks is equivalent to information gained in SLAM through loop-closures, these decorrelation approaches are inherently suboptimal.

In the case of bearing-only measurements, most commonly found in monocular camera SLAM, the linearization of the EKF measurement equations is particularly harsh due to the significant nonlinearity of the perspective camera projection function. As [30] would later show, this effect is made worse because the linearization point changes as the landmark is more accurately estimated, violating the assumption that linearization be about the true landmark location and making the filter inconsistent. [31] partially got around this issue by using an inverse-depth parameterization of landmark locations in the state vector, which approximates a Gaussian distribution more accurately than representing the point as three Euclidean coordinates. However, this parameterization represents landmarks in the state vector as five values, as opposed to the more traditional approach of using the three Euclidean coordinates, worsening the complexity growth. Another approach to this issue is called delayed initialization, in which features are not added to the state vector until they have received sufficiently many measurements [26, 10].

One alternative to Kalman filtering is to use a particle filter, which approximates the posterior probability distribution using particles rather than a Gaussian distribution. A large number of randomly generated particles are propagated through the nonlinear propagation and update equations as an alternative to linearization. The density of the particles represents the posterior probability distribution. The key works bringing the particle filter to SLAM were FastSLAM [32] and its follow-up

FastSLAM 2.0 [33], which gave two key contributions. First, FastSLAM factors the full online SLAM posterior into a product of the path posterior and N landmark posteriors, which reduced the complexity growth to scale logarithmically with the number of features, rather than quadratically. Second, because of the particle representation, FastSLAM can sample over multiple data associations, effectively testing and maintaining many hypotheses. While FastSLAM was originally developed for 2D settings using range-and-bearing measurements, [34] successfully applied this approach to monocular visual SLAM in 3D.

Online SLAM is still an area of ongoing research, including developments such as the Extended Information Filter [35], the Exactly Sparse Delayed-State Filter [36, 37], and the Reduced Dimensionality EKF [38]. However, largely due to the limitations discussed above (i.e., linearization, data association, and computational complexity), the majority of active research is now focused on solving the full SLAM problem.

2.1.3 Full SLAM

The most significant drawback of the online SLAM approach is that new sensor information is not used to improve or correct past results. For this reason, SLAM research began transitioning to the full SLAM problem in the late 1990s, beginning with the Expectation-Maximization (EM) approach of [39, 40] for topological maps and [41] for metric maps. The EM mapping algorithm iterates two steps: an expectation step (which computes the posterior over robot poses for a given map) and a maximization step (which computes the map given the poses), until a local maximum likelihood solution is found. This allows data association to be recomputed as the solution converges, and also for all measurements to be relinearized according to the current best estimate.

Around the same time, the computer vision community was working on two very similar problems to the full SLAM problem. Bundle Adjustment (BA) originated in the photogrammetry community as the problem of computing camera poses and lens distortion using "bundles" of light rays commonly observed in sequences of images [42, 43, 44, 45, 46]. Structure from Motion (SfM), from the computer vision community, is the problem of determining the structure of an object or scene from multiple views [47, 48, 49, 50, 51]. While BA is focused more on optimizing the camera poses and SfM is focused on optimizing the structure, these terms are now used roughly interchangeably, as both approaches typically involve a global optimization for both camera pose and scene structure (though the scene features may be chosen differently).

Parallel Tracking and Mapping (PTAM) reformulated BA to enable real-time calculation of the pose of a handheld camera [52]. This was achieved by separating tracking (localization of the camera in the map) and mapping (global bundle adjustment) into two parallel operations. The mapping component was further accelerated by optimizing only for *key frames*, which allowed camera tracking to run at the full frame rate while mapping ran only when a frame provided significant enough new information to the map. While PTAM had difficulty scaling beyond environments the size of a room, the idea of bundle adjusting only selected key frames became a popular technique in visual SLAM [53, 54, 55, 56]. Davison, et al, provided a detailed comparison of filtering and keyframe bundle adjustment for visual monocular SLAM [57], which showed that keyframe bundle adjustment outperforms the traditional EKF-SLAM filtered approach in nearly every situation.

2.1.4 Graphical Formulation of SLAM

Graphical models have become one of the most popular map representations for the full SLAM problem due to their ease of use and efficiency to solve. An example of a simple pose-graph with odometry, loop-closure, and landmark constraints is shown in Figure 1-2a. In graphical SLAM formulations, the past vehicle poses form the

vertices in the graph, linked by constraints provided by odometry, making up a string of connected poses called a pose-graph. Loop-closures are represented as constraints between non-consecutive poses. In landmark-based SLAM, the landmarks are additionally additionally represented as vertices in the graph, linked to poses by observation constraints. Graphical maps were first used by [58], and were first used with SLAM by [59]. GraphSLAM [60] gave a computationally efficient means of constructing, reducing, and solving SLAM as a smoothing problem. An excellent overview of graphical SLAM is provided in [3], and these techniques have now become widespread [61, 56, 62, 63, 64].

The graphical formulation of SLAM conveniently divides the SLAM problem into that of a front-end, which constructs the graph, and a back-end, which solves the graph. For the purposes of this thesis, the front-end is concerned with generating the odometry constraints (Section 2.2) and detecting loop closures (Section 2.3). The back-end then solves for all of the constraints to generate a maximum likelihood trajectory estimate and environment model. Several generic SLAM constraint solvers have been developed and generally applied [65, 66, 67, 68, 69]. In particular, Incremental Smoothing and Mapping 2 (iSAM2) [68], which incrementally re-orders and solves the sparse graphical model, has been shown to be especially computationally efficient and was selected as the back-end solver for this work.

2.2 Visual Odometry

As described above, localization and SLAM problems are well described by the pose-graph representation, which represents the vehicle and map state as variable nodes and relations between them as factor nodes and edges. Odometry measurements generally provide full-rank constraints between pose nodes in the graph, meaning all degrees of freedom of the poses are constrained. For example, in the 2D case, in which each pose has 3 degrees of freedom, a full-rank odometry measurement constrains both position degrees of freedom (x and y coordinates) and the vehicle heading, θ. The

presence of full-rank constraints between all consecutive poses additionally guarantees "connectedness" of the graph, meaning all degrees of freedom are constrained and the graph can be solved even if no loop-closures are detected. With a high-quality modern odometry system, the vehicle can navigate on the order of kilometers using odometry measurements alone while achieving a suitably low pose error to perform basic routing capabilities [70, 30]. For purposes of background for this thesis, we focus on vision-based odometry systems due to the high quality of recent results, which motivate and enable the contributions of this thesis.

Visual odometry (VO) systems, which estimate the motion of a vehicle using only the input of cameras rigidly attached to it, provide perhaps the best trade between low-cost and high-accuracy odometry estimates of modern, readily available odometry approaches with general applicability. While there are many different VO approaches, the focus is on computing the change in position at the most recent frame. Part of the attraction of VO systems is that they are platform independent, as no knowledge of or integration with the vehicle hardware is required beyond the static transformation between the camera and vehicle coordinate systems (which can be estimated online). Scaramuzza and Fraundorfer give an excellent overview of VO and it's major developments over the past 30 years in their tutorial article [71]. Modern VO systems are very accurate over short durations and smooth, slow motion (typically between 0.1 and 2% relative error at each frame) and are often combined with an IMU to help handle brief periods of rapid motion [72, 73, 30, 74].

Nistér, Naroditsky, and Bergen presented the first real-time implementation of a visual odometry system with robust outlier detection in a calibrated framework [75]. This was achieved by computing frame-to-frame motion estimates using the 5-point relative pose algorithm [76, 77], eliminating outlier features using Random Sample Consensus (RANSAC) [78], and performing a bundle adjustment refinement on the

most recent frames. A large number of subsequent VO developments have been presented [79, 80, 81, 82], leading to reliable systems that can provide high accuracy over long durations, such as 0.1% error on 10 km of rough terrain by [70] or 0.3% error on 21 km of city streets [30].

The current state of the art in visual-inertial odometry is the Multi-State Constraint Kalman Filter version 2.0 (MSCKF 2.0) [30]. The MSCKF is a reformulation of the EKF commonly used for visual odometry and online SLAM such that features are tracked while they remain within the camera field of view and then a single update to the filter state is made when the feature exits the field of view. This requires keeping a sliding window of poses in the filter rather than a single state, but the result is a linearized filter whose observability properties match those of the underlying nonlinear model. Unlike the SLAM formulation of the EKF, the MSCKF is a *consistent* estimator, which means that the estimation errors are zero-mean and the estimation error covariance is greater than or equal to the true error covariance (i.e., the estimator is not overconfident).

As discussed in Section 2.1.4, consecutive poses in a pose-graph are linked using odometry constraints with their associated covariances. The SLAM system then uses the additional information gained from loop-closure measurements to further constrain the system and mitigate the effects of odometry drift. It intuitively follows, and as is discussed in Chapter 3, that the more accurate an odometry system is, the fewer additional constraints in the form of loop-closures and localization measurements are required to keep the state error covariance acceptably low for navigation or other vehicle tasks. Modern visual odometry systems, such as the MSCKF 2.0, are particularly valuable for a SLAM system because they are not only precise and low-drift, but also have unbiased, Gaussian covariance estimates. Using high-accuracy odometry measurements can significantly reduce the importance and value of having frequent

localization or loop-closure constraints for localization or SLAM, respectively. This enables a significant reduction in the size of the localization database, as described in this thesis, which results in increased computational efficiency and reduction in storage requirements.

2.3 Appearance-Based Loop-Closure Detection

As discussed in Section 2.1.4, the graphical formulation of SLAM contains two types of constraints between poses: odometry constraints and loop-closure constraints. Odometry constraints are relative pose measurements between consecutive poses, while loop-closure constraints are relative pose measurements between non-consecutive poses. Whereas Section 2.2 detailed the generation of odometry constraints in a vehicle-agnostic manner from visual and inertial sensors, this section focuses on detecting loop-closures and generating the associated constraints.

The background material presented here focuses on methods operating in the space of visual appearances, whose associated localization and SLAM techniques are commonly referred to as "appearance-based localization" and "appearance-based SLAM," respectively. While any loop-closure method should be compatible with the thesis contributions, appearance-based techniques are here used to illustrate the general process of detecting loop-closures and generating constraints because they have received significant attention in the literature, have been applied to large-scale problems, and tend to be especially data-intensive. Appearance-based techniques are a popular approach in the literature because they use inexpensive cameras as sensors and require no external infrastructure. However, they tend to be data and processing intensive, and as such suffer from a variety of scalability issues. To combat these challenges, a variety of approaches have been introduced to reduce or compress their associated databases, which are described in this section. This thesis introduces a new approach to reducing the storage and processing requirements of loop-closure detection systems, which limits the location database to only the most valuable entries for navigation.

Appearance-based loop-closure techniques apply nearly equivalently to localization as to SLAM. The key difference being that the database of stored locations is predetermined for a localization system, while SLAM systems start from either no or partial prior information to generate, refine, or update the database online, either from scratch or partial information. The contributions of this thesis apply to both localization systems and SLAM systems starting from partial information. The measurements and detection systems for computing "map-relative localization constraints" or "loop-closure constraints" are effectively equivalent, and these terms are used interchangeably in this thesis.

2.3.1 Feature Matching

The simplest means of generating appearance-based loop-closure constraints is by directly matching visual features between images. To do this, unique points in images are detected using a *feature detector* and then described using a *feature descriptor.* Popular visual feature detectors and descriptors include Harris corners [83], the Scale-Invariant Feature Transform (SIFT) [84], Speeded-Up Robust Features (SURF) [85], or Features From Accelerated Segment Test (FAST) [86], which have inspired many follow-on detectors providing various incremental improvements, such as FREAK [87], BRISK [88], ASIFT [89], Gauge-SURF [90], and KAZE [91]. For example, the venerable SIFT descriptor computes a 4×4 grid around the feature and computes the strength of the image gradient in eight orientations, which results in feature vectors with 128 dimensions.

In the case of landmark SLAM, individual features are added into the pose-graph directly as variables, and subsequent matches of a particular feature each generate a new constraint in the graph. In order to limit the total number of constraints that need to be solved, reduce the size of both the pose-graph and location database, and take advantage of a variety of data compression techniques, large-scale systems

tend to favor adding only full-rank loop-closure constraints to the graph. While both approaches are equally valid and applicable to the thesis contributions, the remainder of this section will discuss the latter case of detecting and generating these pose-pose loop-closure measurements.

Once features have been detected and described in all images, they are compared against one another to detect potential loop-closures. If two poses have sufficiently many feature matches (and typically sufficiently few to other poses, as well), they are considered to be viewing the same location and a new constraint can be generated. A number of groups have used this form of matching, perhaps most notably [92, 93] for bundle adjustment of unordered sets of images to reconstruct common tourist attractions and the city of Rome [94].

The most common way to generate the pose constraint is through local structure reconstruction using bundle adjustment. First, the feature correspondences are used to compute the *fundamental matrix*, F (or bifocal tensor), which is a 3×3 matrix that relates homogeneous point correspondences between two images, $\mathbf{x}$ and $\mathbf{x}'$, according to the relationship $\mathbf{x}'^T F \mathbf{x} = 0$ [95]. A number of numerical methods to compute this relationship can be used, such as the 5-point method [76, 77] or 8-point method [96]. Methods such as RANdom SAmple Consensus (RANSAC) [78] are often used to remove any point correspondences that are not consistent with the required epipolar camera geometry, which can filter out transient objects in a scene such as cars.

The relative pose between the two camera views can then be computed from the fundamental matrix, up to a translational scaling factor. The relative pose is used to triangulate the approximate 3D location of the points in space, and this information is used to seed a maximum likelihood bundle adjustment estimation [95]. The result of this bundle adjustment is a relative pose constraint that can be added to the pose-graph. In the case of monocular SLAM, in which the constraint can only be solved up to a scaling factor, additional frames can be added to the bundle adjustment to take advantage of the potentially-known scale of existing odometry measurements.

2.3.2 Bags of Words

The number of pairwise feature descriptor-based matches required to detect a loop-closure measurement grows quadratically in the number of images, quickly becoming intractable for real-time usage. A number of faster alternatives to pairwise matching have been proposed, including multi-round matching schemes [94] and tree-based indexing [97].

One particularly common method of speeding up these comparisons is to use the *bag-of-words* representation, which enables a visual analogy to textual document retrieval approaches used by web search engines [98]. Text retrieval systems first parse documents into words, represent these words by their stems (e.g., 'search' rather than 'searching'), and assign each word a unique identifier. Each document can then be represented by a vector of word frequencies and a vector of word weights. The standard weighting method is called "term frequency–inverse document frequency" (tf-idf) [99]. If the vocabulary contains k total words, each document is represented by a vector of weighted word frequencies $\boldsymbol{V}_d = (t_1, ..., t_i, ..., t_k)^T$ with weights computed as:

$$t_i = \frac{n_{id}}{n_d} log \frac{N}{n_i} \tag{2.1}$$

where N is the total number of documents, n_{id} is the number occurrences of word i in document d, n_d is the total number of words in document d, and n_i is the total number of occurrences of word i in the entire collection of documents [100]. The tf-idf weighting is the product of the word frequency, which values repetitive words in a particular document, and the inverse document frequency, which devalues words appearing frequently in the document collection, making it an effective measure of information content in the context of document search. This process must be carried out prior to any searches being conducted, as both the dictionary of all possible

word stems and their occurrences must be pre-computed using the full collection of documents. The vectors representing the documents can then be organized using an inverted index, which maps each word to a list of documents it appears in, enabling fast document look-up [101].

Sivic and Zisserman [100] applied these concepts from document search to visual image search. Feature descriptors detected in images are used in place of textual words, using a vector quantization scheme to find the "feature" stem. An image can then be represented using a vector of visual word occurrences and a weighting vector, which is called *bag-of-words* representation. This has the effect of compressing the image down to a vector of visual words weighted by their information content in the context of image matching. The use of an inverted index effectively means that image matches are largely pre-computed, greatly speeding up visual search for image matching and object detection. Visual bag-of-words-based object matching has been successfully applied to various generic recognition problems, such as finding multiple movie frames containing a specific object [102] and visual search using a camera phone [103].

The combination of the information-weighted words with the inverted index makes the bag-of-words representation both robust and computationally efficient, making it well suited for application to appearance-based loop-closure detection. [102] applied bag-of-words matching to loop-detection for SLAM, and showed that better retrieval quality is obtained by increasing the vocabulary size. In order to speed up performance when using a large vocabulary, they used hierarchically defined visual words that formed a vocabulary tree, allowing more efficient word lookup using a decision tree. They also quantized the space of *all* possible visual words (using SIFT descriptors), so that once the vocabulary has been learned offline, new images can be inserted into the database on-the-fly rather than requiring an *a priori* learning process. [104] learned the vocabulary incrementally during operation by maintaining a separate "young word" database containing candidate words for addition to the main vocabulary, showing performance nearly equivalent to vocabularies learned offline. [105]

introduced "dynamic bag-of-words," which finds local sets of observed landmarks at query time rather than beforehand by utilizing co-visibility (features that are observed together). [106] showed improved bag-of-words reliability by grouping nearby images into "islands" to prevent nearby images from competing with one another. [107] showed a comprehensive appearance-based SLAM system using bag-of-words for place recognition. Recent work has focused on extending bag-of-words approaches to recognize places whose appearances change over time [108]. For example, Churchill and Newman represent each location in the environment as a set of "experiences," which can represent distinct weather or lighting conditions [109, 110, 111].

Bag-of-words matching originated in the visual object detection and recognition community, in which the goal was to determine the classification of an object according to some set of object categories (bikes, cars, etc.). It is well known in the computer vision community that bag-of-words matching performance is highly correlated with the quality of the vocabulary (and thus the weights used), which is a function of the number of matching categories and the similarity of the sets of images used for training versus matching. Therefore, it is important to carefully select the set of images used to learn the dictionary such that the training images sufficiently span the space of visual variation present in the set. For example, bag-of-words matching has been shown to work well for object recognition for objects with both visually-dissimilar categories, such as those in the PASCAL Challenge (cars, cows, bikes, etc.) [112], and visually-similar categories, such as species of flowers [113], so long as the training images are sufficiently representative of the dataset. This can be a challenge for SLAM applications, which essentially treat every pose as a separate recognition category without any designated training images for specific categories (i.e., specific vehicle poses), and thus must use very large vocabularies. While left for future work, Section 7.1.2 briefly describes how the location selection approach presented in this thesis might enable better bag-of-words vocabulary learning approaches by allowing physical locations in space to be treated as recognition categories with multiple, designated training images, which may lead to better matching performance and reliability.

2.3.3 Fast Appearance-Based Mapping

Fast Appearance-Based Mapping (FAB-MAP) [13] reformulated bag-of-words by using a generative model of place appearance, which allows a probability of recognition to be computed for each previous image, rather than the tf-idf metric (2.1). Identical but indistinctive observations (such as those of the most common scenes) are given a comparatively lower probability of coming from the same place, reducing the effects of *perceptual aliasing* (i.e., different locations appearing the same). The system learns a generative model of the bag-of-words data, which accounts for certain combinations of visual words tending to co-occur due to being *generated* by common objects. The result is that places can be recognized even with few feature matches, while rejecting false matches due to perceptual aliasing.

FAB-MAP 2.0 [114] extended FAB-MAP, making it scalable to very large trajectories by defining a sparse approximation to the FAB-MAP model that allowed implementation using an inverted index. Geometric verification is performed on the top 100 most likely locations using RANSAC as an added robustness check. FAB-MAP 2.0 performed appearance-based loop-detection on trajectories up to 1,000 km long with computation time growing linearly with distance traveled.

An open-source implementation, OpenFABMAP [115], is used by many SLAM systems, including [108, 62, 61, 116], and its frequent use as a reference for comparison [117, 118] has established FAB-MAP as a state-of-the-art appearance-based loop-closure detection system. However, FAB-MAP is not without drawbacks. While FAB-MAP 2.0 did manage to close loops on a 1000 km trajectory, the results from a 70 km trajectory were much better, as shown in Table 2.1. For example, at the 100% precision level (no false-positive loop closures), the 70 km dataset achieved a 48.5% recall, while the 1000 km dataset only achieved 3.1% recall. Recall is the fraction of location revisits for which *any* loop closure was detected, and precision is the percentage of *correct* loop-closures. Generally loop-closure detection systems aim to maximize recall while maintaining 100% precision, as incorrect loop-closures are

difficult to recover from and can prevent a pose-graph solver from finding the correct solution. Additionally, the vocabulary in FAB-MAP must be learned offline prior to operation, and the training data used can greatly affect recognition performance (i.e., generic vocabularies cannot be used).

One of the potential applications of this thesis is to reduce the appearance data required to navigate extremely long trajectories, such as the 1000 km dataset used to test FAB-MAP 2.0, such that they achieve similar results and data storage as a much smaller dataset, like the 70 km trajectory, by predetermining the most valuable locations for loop-closures to occur and ignoring all others. For example, if a location is unlikely to be re-visited or recognized, it should not be stored in the database. The case of making such determinations online, as would be required to improve upon FAB-MAP's results, has been left for future work and is discussed further in Sections 7.1.1 and 7.1.2.

Data set	70 km	1000 km
Number of Loop Closures	4,757	48,493
Extracted Features	16 GB	177 GB
Recall (100% Precision, 100k words)	48.5	3.1
Recall (99% Precision, 100k words)	73.2	8.3
Recall (99% Precision, 10k words)	52.3	2.7

Table 2.1: A comparison of FAB-MAP 2.0 results from 70 km and 1,000 km datasets in similar environments [114]. Recall is the fraction of location revisits for which *any* loop closure was detected, and precision is the percentage of *correct* loop-closures. Notice that recall dropped by about an order of magnitude when the trajectory length grew by about an order of magnitude.

2.4 Map Reduction

In the canonical form of graphical SLAM, a new pose node and odometry constraint are added to the map at every time step, causing the map to grow at least linearly in time. Additional loop-closure constraints are added whenever loop detections occur. This means that a typical graphical map will add poses linearly and constraints *at least* linearly in time.

Furthermore, to enable loop closure, the system must store sufficient data for data association at each previous pose. For visual appearance-based loop-closure detection, discussed in Section 2.3, even when using the efficient bag-of-words compression with a pre-learned dictionary, this requires storing a vector of weighted visual words for each node. As an example, if a system adds nodes at 1 Hz, uses a 10,000-word dictionary, and stores the weighted vector of visual words as 16-bit integers, it will generate nearly 70 MB of data per hour.

Map growth leads to several major challenges in localization and SLAM. First, the pose-graph becomes more difficult to perform inference on as additional poses and variables are added to the graph, increasing the dimensionality of the optimization problem. Second, adding too many loop-closure constraints can cause the graph to lose the sparsity exploited by computationally efficient back-end pose-graph solvers [67, 66, 35, 36, 119]. And third, both data association and data storage become increasingly difficult as data volume grows due to processing requirements and location aliasing.

The remainder of this section provides a review of the four general approaches used to manage map growth in SLAM. The first approach, called submapping, divides the map into smaller portions that can be updated in real-time. Graph sparsification approaches attempt to remove uninformative constraints and poses from the pose-graph in the back-end solver after the graph has been constructed. Constraint selection approaches operate instead in the front-end, passing only sufficiently informative constraints to the solver that are likely to reduce the map and trajectory covariances,

again requiring access to the pose-graph. The final class of map reduction approaches includes various strategies to bound the growth of the map, such as by the explored area or operating time. This thesis introduces a new approach for map reduction in the form of an optimal sensor placement problem, reducing the map prior to vehicle navigation without access to the vehicle trajectory. Section 2.4.5 provides background on the optimal sensor placement problem, which typically places a limited number of sensors to gather information within a region.

2.4.1 Submaps

As discussed in Section 2.1.2, EKF-SLAM has significant difficulty dealing with large maps, as the entire map is stored in the state vector and EKFs must invert the state covariance matrix at every time step. To enable SLAM in larger environments without this quadratic growth in computation, techniques such as submaps and hierarchical maps were developed [28, 29, 120]. These methods involved maintaining multiple maps that could be swapped in or out of the state as different regions of the environment were traversed. However, these systems are not able to take full advantage of loop closures due to the decorrelation between the submaps required to maintain global consistency. Similar methods were used to extend the range of bundle adjusted visual SLAM systems through use of multiple local maps [121] or limiting bundle adjustment to only an active, proximal region of the map [62, 122]. Tectonic SAM [123] applied the idea of local submaps to graphical SLAM, with local submaps being optimized independently in local coordinate frames before being combined into a global map. However, while these submapping methods bound computational growth, they do not reduce the total data storage required for data association.

In graph theory, the Connected Dominating Set (CDS) of a graph is the minimal subset of vertices such that every vertex in the original set is either in or neighboring a vertex in the dominating set. While determining the CDS is NP-complete, good approximations are available in polynomial time [124]. This concept has been applied to reduce appearance-based maps to the minimal subset of images that completely

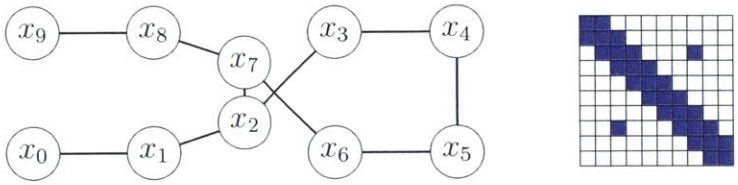
cover a particular area [125, 126]. This guarantees that any new image taken within the mapped region will overlap with at least one image existing in the map, allowing localization on a map bounded by area. [54] used a variation of this concept, which they called a "skeletal graph," to speed up large-scale bundle adjustment problems. They first form and bundle adjust the skeletal graph, then use that result to initialize the full bundle adjustment problem, enabling the famous reconstruction of the city of Rome in one day of computation [94]. [107] applied the concept of a skeletal graph to SLAM, retaining only a skeletal graph in areas of dense exploration by removing nodes online to control the density of appearance-based data in the map.

2.4.2 Graph Sparsification

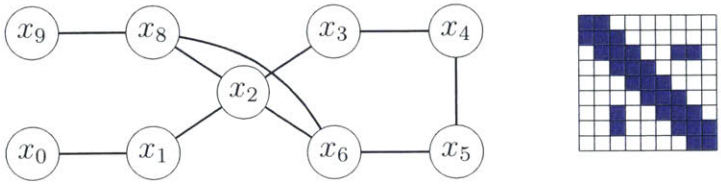
Pose-graph SLAM entails solving a large least-squares problem over all poses. To do so efficiently, most solvers judiciously exploit sparsity in the problem structure [67, 66, 35, 36, 119]. This sparsity results from odometry constraints affecting only consecutive poses and loop-closures being occasional and limited. In some cases, such as long-duration, multi-session systems, the graphs grow too large to solve in real-time. To mitigate this issue, pose-graph sparsification algorithms can be used to reduce the number of poses and constraints in a map, thus improving solver efficiency. These algorithms work exclusively in the pose-graph solver, or "back-end."

The sparsest possible pose-graph (without any breaks) is one with no loop closure constraints. As more loop closure constraints are added to the graph it becomes more dense, and the system becomes more difficult to solve. Additionally, when vertices are marginalized out (i.e., removed from the graph), additional edges appear in the graph in their place, as shown in Figure 2-2, making the graph more dense.

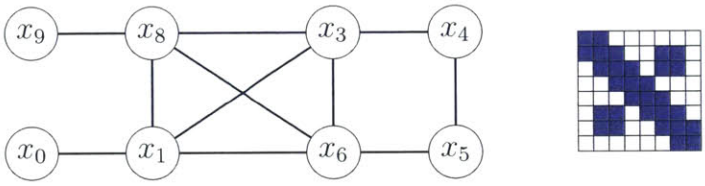
To enforce sparsity, some have proposed simply removing the weakest constraints [107, 127], but this can cause the system to become biased and inconsistent. Carlevaris-Bianco et al. [128, 129, 130, 131] consolidate densely connected regions of a pose-graph into Generic Linear Constraints (GLCs), while Huang et al. [132] use ℓ_1-optimization



(a) Original graph: 30% fill-in.



(b) Pose x_7 removed: 36% fill-in.



(c) Poses x_7 and x_2 removed: 47% fill-in

Figure 2-2: Pose removal through marginalization adds density to the connectivity matrix. (a) shows a pose-graph with two poses located very near to one another. (b) shows the effect of combining these poses, and (c) shows the effect of removing them. Both of these graph reduction operations reduce sparsity.

to consistently remove weak edges in the graph. Sparsification approaches have been used in conjunction with location saliency metrics [133] in order to weight the optimization towards retaining the database entries that are most likely to be recognized [134], but do not explicitly incorporate the likelihood of revisit.

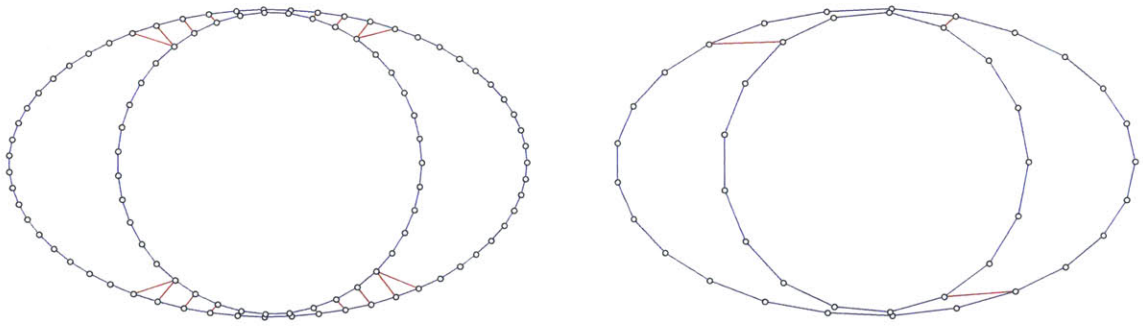
All graph sparsification approaches have involved removing or replacing constraints after they have already been added to the graph. In contrast, the database selection approach presented in this thesis does not require access to the full graph when determining which *predicted* constraints to enable through a limited location database.

2.4.3 Constraint Selection

Others have taken an information-theoretic approach to reducing map size and promoting sparsity in the front-end [135, 136]. [120, 137] showed how SLAM could be considered in an information theoretic framework, investigating the *information* encoded in measurements and estimates. Kretzschmar et al. [138, 139] estimated the mutual information of laser-scan measurements with regard to an occupancy grid, only incorporating new measurements when they were sufficiently informative. They marginalize out old poses to bound map growth to a fixed memory size.

Information-Based Compact Pose SLAM [4] generalized these ideas to pose-graphs, showing that by keeping only highly-informative loop-closure constraints and non-redundant poses, the pose-graph can be solved not only faster than with dense poses and constraints, but also more consistently. They introduced a method to compute the information content of a constraint online in constant time, and used this to determine whether or not to add the constraint to the graph. The result was a system that closed few loops and operated in open-loop on odometry for long periods, as shown in Figure 2-3. Additionally, by only adding informative odometry constraints, the result was similar to a keyframe-based system, such as FrameSLAM [53], in which the keyframes have equal spacing in *information-space* rather than *Euclidean-space*. [140] approximated the robot trajectory using a sequence of straight line segments, achieving a roughly similar odometry result to [4]. Wang et al. [141] pruned poses using a measure based on the Kullback-Leibler divergence between a given frame and the map in feature-based SLAM, rather than mutual information.

Kim and Eustice [133] pointed out that appearance data is only valuable if it leads to a future loop-closure, and that the "saliency" (distinctiveness) of images corresponds to how likely they are to be both re-detected and capable of generating a relative constraint. To measure visual saliency, they introduced two metrics for the bag-of-words representation of an image: global saliency (the rarity of an image) and local saliency (the texture richness in an image, and thus registrability). Global



(a) Standard pose-graph. (b) Reduced pose-graph of [4].

Figure 2-3: Information-based pose-graph reduction. Constraints are only added to the graph when they contain sufficient information content, resulting in a significant reduction in map size.

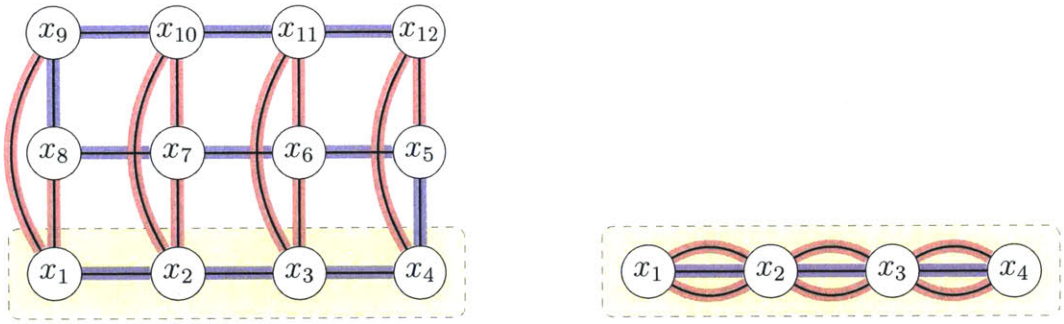
saliency is related to the inverse document frequency (2.1), while local saliency is defined as the normalized entropy of the bag-of-words frequency vector. They found that images with low local or global saliency are unlikely to result in appearance-based link hypotheses and can be safely discarded, reducing computation and storage requirements. They then used these saliency scores to select vehicle trajectories for environment exploration and area coverage that would result in sufficiently many loop-closures in visually salient regions of the environment to correct for odometry drift and maintain vehicle localization performance [142, 143].

While effective at map *constraint* compression, all of these approaches require access to the sensor data and typically first compute the actual constraint before determining whether or not to incorporate it into the map. In comparison, the map reduction approach presented in this thesis does not have access to the actual sensor measurements or constraints, and instead must evaluate the value of *potential* constraints. Furthermore, in order to enable future loop-closure constraints, these constraint selection approaches maintain full location databases even when constraints are not added to the graph, thus the databases still grow continuously in time.

2.4.4 Bounded Graphs

Rather than focusing explicitly on map sparsification, other approaches have sought to bound map growth. As discussed above, canonical graphical SLAM systems have maps that grow at least linearly in time. However, clearly a biological brain does not increase in size as it explores its environment. RatSLAM [16, 144] is a SLAM algorithm inspired by computational models of the hippocampus of rodents, which were found to display many properties of a desirable robotic SLAM system. RatSLAM uses *pose cells* rather than a Cartesian coordinate system, which represent experiences, with each of these cells representing a place in memory with appearance-based data. [145] extended RatSLAM to allow experience consolidation for long-term operation in dynamic environments. By overlaying a grid over RatSLAM's experience map, they limited the system's memory to only one experience per grid location. Given the assumption that the majority of the environment is static, this allows dynamic experiences in various locations while the map size remains bounded by area. Because the experience map is topological rather than metric, the connections between experiences can be simply rerouted within the grid. [145] demonstrated operation of a robot performing mock deliveries in a working office environment continuously over a two week period using RatSLAM.

The Reduced Pose Graph [146] applied a similar concept to metric visual SLAM, effectively placing a grid over the world in cartesian space and enforcing that only one pose vertex can be located in any grid cell, and bounded the number of poses (and thus the size of the location database) by area rather than time. When a location was revisited, constraints were added only to existing poses rather than creating new ones, bounding the number of poses (and thus the size of the location database) by area rather than time, as shown in Figure 2-4. Because constraints were added only to existing poses in explored regions, sparsity was maintained and no marginalization



(a) Standard pose-graph.

(b) Reduced pose-graph of [146].

Figure 2-4: Area-based pose-graph reduction for three traversals over a region, shown in yellow (i.e., $x_8 \approx x_1$). Instead of adding additional poses to the graph, additional constraints are added between existing poses.

was used to reduce the graph. The system retained multiple constraints and combined them when consensus was reached in order to avoid incorporating bad constraints into a combined edge, and was demonstrated in a complex, multi-floor environment on nine hours of data.

Rather than bounding map growth in area, Real-Time Appearance-Based Mapping (RTAB-Map) [147] bounded the map by computation required for search by using a memory management method that limited the number of locations available for loop-closure detection. They divided the location database into active and inactive locations, with only the active locations being searched for loop closure, and the inactive locations being promoted to the active set when a nearby active location was visited. RTAB-Map can handle large-scale, long-term operation, but if a location is not revisited sufficiently frequently it will be forever lost to long-term memory.

The Reduced Pose Graph [146] successfully bounds map growth by area and RTAB-Map [147] bounds the computation required for map search during data association. However, like the sparsification and constraint selection methods, these approaches require access to the actual measurements in order to reduce the map. The reduction approach presented in this thesis instead limits the map to the available storage size, effectively selecting the most valuable map components that will fit within storage constraints without requiring access to the sensor measurements or pose-graph.

2.4.5 Optimal Sensor Placement

In determining which locations to close loops, this thesis introduces a new optimal sensor placement problem in the context of vehicle localization. Each database entry can be considered a navigation beacon, and the problem becomes one of selecting the optimal set of beacons about the environment. Optimal sensor placement generally entails minimizing the number of sensors while maximizing coverage. The utility-based selection approach presented in this thesis instead selects the N best sensors given a predefined set (all traversable locations) to minimize expected trajectory uncertainty rather than maximize coverage.

Krause et al. formulate an optimal sensor problem by placing sensors to maximize the mutual information of sensed and unsensed locations in an environment using learned Gaussian processes [148]. Beinhofer et al. [149] and Allen et al. [150] formulate sensor placement problems to place artificial navigation landmarks and sonar beacons, respectively, along pre-defined trajectories for localization. Vitus and Tomlin [151] formulated an optimal sensor placement problem in which a vehicle deploys a limited set of sensor beacons in the environment to minimize its navigation estimation error.

Optimal sensor placement problems can additionally be formulated to select minimal sets of natural landmarks in the environment to be incorporated into the vehicle's map. Frintrop and Jensfelt [152] used active camera gaze control to track a limited set of features selected to achieve good spatial distribution throughout the map. Hochdorfer and Schlegel [153] selected landmarks to maximally cover the vehicle's operational environment given bounded resources. Zhang et al. [154] instead used an entropy metric when determining which landmarks to add to the map, considering the uncertainty reduction in the vehicle's position resulting from incorporation of any additional landmark. Dissanayake et al. [155] and Strasdat et al. [156] introduced methods to remove low-value landmarks from the map in order to improve computational efficiency.

The location database selection approach presented in this thesis involves choosing a limited-size subset of the locations in the environment prior to navigation at which potential loop-closure measurements will be most beneficial to *any* trajectory through the environment. This is distinct from other optimal sensor placement problems in the context of vehicle localization because the specific trajectory and measurements are unknown, and we instead must develop and rely on a probabilistic representation of the vehicle trajectory.

2.5 Path Planning

One of the key distinguishing factors of the map reduction approach presented in this thesis is that the reduction occurs prior to vehicle operation. This means that, rather than having observability of the true vehicle trajectory, we must develop a probabilistic representation of the vehicle trajectory, which will then be used to select valuable locations in the environment. This is accomplished in part by exploiting the structure of vehicle motion and the environment through knowledge or observation of the vehicle's path planner, as will be described in depth in the following chapter.

The value of the map to the larger system is primarily to enable the guidance system to select optimal, traversable paths. Therefore, it is essential that a reduced map to be used for vehicle guidance retains the value of the map to the planning system. However, SLAM representations have traditionally been designed without considering path planning performance. A great wealth of graph-based algorithms already exist from the graph theory community [157, 158], and those most applicable to path planning on graph-based maps and the utility-based map reduction approach are discussed here. To do so, it is important to first formally define a *graph*.

Definition 1. *A graph is denoted by* $G = (V, E)$*, where* V *is a finite set of vertices and* $E \subseteq V \times V$ *is a set of edges.*

For simplicity of notation, we denote the number of vertices as $|V| = n$ and the number of edges as $|E| = m$.

Before performing graph-based path planning, it is important to first consider the *reachability* of a graph. The reachability relation refers to the connectivity of two graph vertices by edges. If a graph satisfies the reachability relation, then a path can be computed between any of its vertices. This is an important requirement for the algorithms for path planning on graphs given below.

Definition 2. *Graph G satisfies the reachability relation if for the set of all ordered pairs (s, t) of vertices in V there exists a sequence of vertices $v_0 = s, v_1, v_2, \ldots, v_k = t$ such that edge (v_{i-1}, v_i) is in E for all $1 \leq i \leq k$.*

For the specific case of pose-graphs, the graph exhibits the reachability relation as long as the system produces an odometry constraint between all consecutive poses. This can be guaranteed by using a fully internal odometry sensor with no risk of outages, such as an inertial measurement unit (IMU). It is important that any *reduced* graph retain this property if it will be used for graph-based path planning.

2.5.1 Shortest Path

For a graph-based map, the optimal path, $p = (p_1, p_2, \ldots, p_k)$, between two vertices $(s, t) \subseteq V$ is defined as the path minimizing some cost function, also called the *shortest path*. The edges can either all be equivalent, and thus the shortest path is that which traverses the fewest edges, or weighted, such as by distances or travel times. An example of a weighted graph is one in which the vertices are locations in the world and the edges are the roads between them, weighted by their lengths.

Breadth-first search (BFS) [158] can be used to find the shortest path on a graph with equal edge lengths in linear time, $\mathcal{O}(m)$. BFS starts from vertex s and first searches all vertices directly connected to it for the goal vertex, t. If t is not found, the process repeats for each vertex one edge away from s. If t is again not found, the process repeats for increasingly large numbers of edges. This is in contrast to the *depth-first search*, which instead travels as far as possible from the initial vertex whenever possible.

Graphs with non-negative edge lengths are handled by adding a length function, $\ell : E \to \Re_{\geq 0}$ and defining the optimal path, p, as that which starts from s and reaches t while minimizing $\sum_{j=1}^{k} \ell(e_j)$. *Dijkstra's algorithm* [159] generalizes BFS for weighted graphs with non-negative edge lengths, such as pose-graphs, in $\mathcal{O}(n^2)$ time. The algorithm maintains two data structures, $d : V \to \Re_{\geq 0}$, the estimate of the distance from s to all other vertices, and $U \subseteq V$, the set of "undetermined vertices." Subsequent modifications to Dijkstra's algorithm have managed to reduce the time complexity, such as through use of Fibonacci heaps to $\mathcal{O}(m + n \log n)$ [160] and radix heaps for integer weights to $\mathcal{O}(m + n\sqrt{\log C})$ [161]. An example of Dijkstra's algorithm on a simple graph is shown in Figure 2-5.

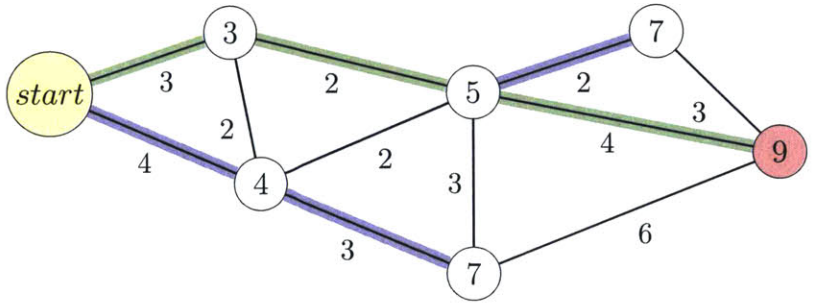


Figure 2-5: Dijkstra's algorithm finds the shortest path from a selected vertex to all other vertices in a weighted graph using breadth-first search. The distance to each vertex is shown, and every edge traversed by the algorithm is highlighted. The shortest path to the goal node (shown in red) is highlighted in green.

Dijkstra's algorithm was extended using heuristics to the A* search algorithm [162], which is called a *best-first search.* A* search uses a knowledge-plus-heuristic cost function of vertex x: $f(x) = g(x) + h(x)$, in which $g(x)$ is the known distance from s to x, and $h(x)$ is an estimate of the distance from x to t. The edge with the lowest cost $f(x)$ is searched first at every time step. As long as $h(x)$ serves as a lower bound for this distance (i.e., $h(x) \leq d(x, y) + h(y)$ for every edge (x, y)), then A* returns the shortest path while expanding the fewest possible vertices.

2.5.2 All-Pairs Shortest Paths

The all-pairs shortest path problem expands the single-pair shortest path problem to every pair of vertices in the graph. The result is a matrix of all distances between pairs of vertices, called the *graph distance matrix.*

The simplest approach to calculating the graph distance matrix is to iterate the single-pair algorithms, described in Section 2.5.1, n times (for every pair of vertices). For Dijkstra's algorithm, this raises the complexity to $\mathcal{O}(n^3)$ (and similarly multiplies the complexity of the other above algorithms by n). Note that Dijkstra's algorithm computes the distance from s to all other vertices in $\mathcal{O}(n^2)$ when not terminated after reaching t.

The Floyd-Warshall algorithm [163] also computes the shortest paths between all pairs of vertices in a weighted graph in $\mathcal{O}(n^3)$, but additionally allows negative edge weights (assuming no negative cycles). *Johnson's algorithm* [164] similarly computes the shortest paths between all pairs of vertices for weighted graphs with some negative weights (but no negative cycles), but can additionally handle directed graphs. Its worst-case performance is $\mathcal{O}(n^2 \log n + nm)$ (when implemented using Fibonacci heaps), which outperforms the Dijkstra and Floyd-Warshall algorithms for sparse graphs.

2.5.3 Path Planning with Graphical SLAM

Objective-seeking path planning algorithms require a map of the environment. When an *a priori* map is unavailable, the system must generate this map online, which will involve SLAM unless an external localization system is available and sufficient (e.g., GPS). Typically, path planning algorithms require either a map of obstacles, such as a building floor plan, or a map of traversable paths, such as a road map. Traversability maps enable more efficient planning, but require an additional system for obstacle detection. Obstacle maps enable continuous-domain trajectory planning, but the algorithms for this typically come at a higher computational cost and employ techniques such as sampling for complexity reduction. Examples of popular sampling-based planning algorithms are probabilistic roadmaps (PRM) [165] and rapidly-exploring random trees (RRT) [166, 167].

A number of systems have included both path planning algorithms onboard and SLAM, though most early systems used algorithms and representations designed somewhat in isolation. Examples of such vehicles include autonomous cars from the DARPA Grand Challenge [168, 169], Autonomous Underwater Vehicles (AUVs) [170], and Unmanned Aerial Vehicles (UAVs) [171, 172, 173]. However, self-driving cars require huge computational resources, and systems used onboard small quadrotors have been limited largely to downscoped algorithms such as waypoint guidance [172] or off-vehicle map processing [173]. As opposed to objective-based path planning, similar algorithms have also been employed to plan exploration trajectories that maximize mapping coverage of an environment. For example, AUVs have successfully employed PRM and RRT for complete-coverage ship-hull inspection [174].

In an attempt to make path planning algorithms scale better to large environments, [17] introduced a hybrid metric-topological map. This SLAM representation was developed specifically for efficient large-scale path planning and used a hierarchical map structure, similar to those described in Section 2.4.1, with detailed local metric submaps and a global topological map of connectivity. [17] argued that topolog-

ical connections with approximate metric information were sufficient for planning over large spaces. They used Dijkstra's algorithm to compute the shortest path to the goal through the topological graph and constructed occupancy grids within each of the submaps the trajectory would cross. They then used these occupancy grids for sampling-based path planning for obstacle avoidance in continuous space, using the topological vertices effectively as waypoints. [175] applied a similar approach to navigating a vehicle in a crowded and dynamic city environment, planning global trajectories using a topological representation and rendering detailed maps only locally. Their results agreed with those of [17], finding that their planned trajectories were about 10% longer than optimal but were computed several orders of magnitude faster than when using a single metric map. They additionally used a mid-level representation based on the pose-graph vertices to compute waypoints using Dijkstra's algorithm, which are sent to the low-level planner. [176] extended [17] by caching the local maps for greater efficiency in multi-agent scenarios.

A vehicle navigating using SLAM will almost always have uncertainty in both the map and the position, and often this uncertainty is significant. [177] introduced belief roadmaps (BRM), which extended PRMs to *belief space*, or the space of probabilistic position estimates. They showed improved vehicle motion performance by incorporating the predicted uncertainty of future position estimates into path planning, and demonstrated this on a quadrotor using a laser range-finder [178]. [179, 180] claimed that because feature-based SLAM produces a sparse map, it cannot be used directly for path planning because very little obstacle information is available. However, they showed that a pose-graph can be used directly as a BRM because odometry constraints represent traversability and the information content of measurements at pose nodes represent the information available in the environment. They then gave a method to determine an optimal navigation strategy to reach a goal location by searching the pose graph for paths with low accumulated pose uncertainty. [181] extended this work by incorporating active exploration to balance between map building and revisiting known locations. This system guided the vehicle to maximize coverage while minimizing both localization and mapping uncertainties. [182] similarly found

minimum uncertainty trajectories on generic pose-graphs, but introduced a reduced graph representation called a *decision graph* to simplify path planning using Dijkstra's algorithm. The decision graph retains only the start and goal vertices and the vertices of the pose graph with degree greater than two, as only these locations involve decision making.

The majority of path planning algorithms applied to SLAM have focused on maximizing map coverage [183], often treating environment exploration as a form of the "traveling salesman" problem, or operating within existing maps [171]. However, in many situations in operational robotics, the vehicle is solely trying to reach a goal. Assuming all mapped regions are reachable, exploration is only *necessary* when the goal lies in an unexplored region of the map. A highly risk-averse vehicle sees no value in unnecessary environment mapping. An example of an algorithm designed to minimize localization uncertainty during exploration is the Network of Reusable Paths [184, 185], which selected vehicle trajectories such that vehicle remained on previously traversed paths whenever possible. When the goal location lied outside of the network of paths, an exploration path was planned to minimize the expected localization error of the vehicle at the goal. However, the Network of Reusable Paths was formulated only to minimize localization error at the goal, rather than to maximize the information content of a map.

Chapter 3

Location Utility

Localization and SLAM systems seek to minimize the error of a vehicle trajectory estimate (and map estimates, in the case of SLAM) using measurements provided by navigation sensors. For purposes of this chapter, we will assume that this measurement set contains some form of odometry and loop-closure measurements in a pose-graph formulation (see Section 2.1.4) and all consecutive poses are connected by full-rank odometry measurements (see Section 2.2). However, the loop-closure measurements discussed in this chapter could be replaced by an absolute or map-relative measurement, such as from a GPS receiver or correlation to an existing map.

Our goal is to identify the most useful locations in a map for loop-closure or absolute measurements to occur, given some basic prior information about the map and expected trajectory. This goal is more rigorously defined in Section 3.1, which motivates the development of a metric of location utility to facilitate selecting locations according to their value for potential loop-closure measurements in Chapter 4. Section 3.2 introduces the location utility metric for the case of a planar network of traversable paths with full-rank measurements. Section 3.3 then extends the definition to cases with unrestricted 6-degree-of-freedom motion and low-rank projective landmark measurements.

3.1 Reducing Pose Uncertainty

The goal of vehicle localization is to determine the vehicle's pose as accurately as possible. Assuming that all measurements are accurate and their uncertainty is well-modeled, this goal is achieved in practice by minimizing the vehicle pose error uncertainty, which determines the precision of the solution. It is well known that the pose error uncertainty will grow unbounded in time when using solely odometry measurements and can only be reduced by incorporating either an absolute or loop-closure constraint. These uncertainties are represented as covariances when the posterior of the vehicle poses is represented using a multi-variate Gaussian distribution, as is common in the literature [3, 67, 66]. The primary goal of a navigational localization or SLAM system is therefore to gather and apply absolute or loop-closure constraints to minimize the vehicle pose error covariances of the trajectory.

The trajectory uncertainty can be quantified by the average pose position uncertainty, ε, of all N poses in the trajectory:

$$\sigma_{pos} = \sqrt{\mathrm{Tr}(\mathrm{Q_{pos}})} \tag{3.1a}$$

$$\varepsilon = \sum_{n=1}^{N} \frac{\sigma_{pos,n}}{N} \tag{3.1b}$$

where Q_{pos} is the marginal position error covariance matrix of a pose, Tr is the matrix trace operator, and σ_{pos} is the uncertainty of an individual pose. A marginal covariance represents the uncertainties between a subset of variables (e.g., the x and y positions of a particular pose) as opposed to all variables in the problem [186]. We neglect attitude uncertainty to avoid the unit conflict between meters and radians, but a metric of average pose attitude uncertainty could be similarly defined, if desired. Position and heading uncertainty are correlated in cases in which all consecutive poses are connected with full-rank odometry constraints, as we assume in this chapter, making it sufficient for the purposes of our analysis to quantify trajectory uncertainty using solely the pose position uncertainty.

The pose error covariance in a pose-graph based localization or SLAM system is ultimately a function of the following three factors:

1. the number of loop-closure measurements,
2. the distribution of loop-closure measurements, and
3. the individual measurement covariances.

The configuration that minimizes the pose error covariance will have the maximum number of maximally-distributed, maximally-confident measurements. Put more simply, we would like to have many well-distributed, precise measurements.

Figure 3-1 shows a simple 2D pose-graph of a vehicle moving one unit forward at each time step, consisting of a string of poses, x_i, connected by identical odometry measurements. Each pose additionally has an optional full-rank prior, z_i, constraining its location and heading in absolute space, each of which represents a loop-closure measurement to a well-estimated pose.

Figure 3-2 shows the error covariances of this pose-graph, given various combinations of loop-closure constraints, z (i.e., these constraints are switched on or off). If we assume that all loop-closure measurements are equally precise, the two remaining sources of map covariance variation are the number and spatial distribution of loop-closure measurements. The average pose covariance is reduced as additional measurements are added (see Figure 3-2a), and, for a fixed number of constraints, the average pose covariance is minimized when the constraints are maximally distributed (see Figure 3-2b). Figure 3-3 shows the optimal measurement configuration of those evaluated in Figure 3-2, which minimizes the average pose position error by maximizing the allowable measurements (three) and their distribution, ε.

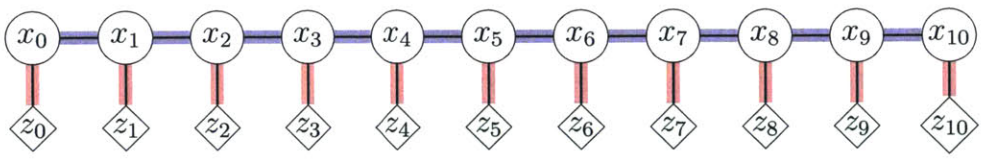
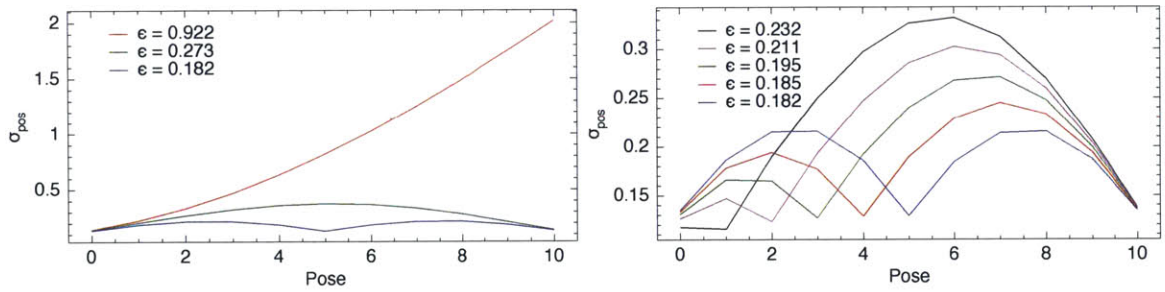


Figure 3-1: A simple graph of eleven poses, x_i, connected by odometry measurements (blue). A measurement to the external frame, z_i, is additionally available at each pose, which constrains its location in the global coordinate frame.



(a) (b)

Figure 3-2: Pose position uncertainty, σ_{pos}, for poses in the simple trajectory shown in Figure 3-1 with a variety of measurement configurations. Figure (a) shows the position uncertainty when optimizing the pose-graph with one (red), two (green), and three (blue) global-frame measurements. Figure (b) shows the position uncertainty when optimizing the pose-graph while varying the pose associated with the third global-frame measurement. The configuration with minimal pose error (blue) maximizes both the number of measurements (left) and their spatial distribution (right) and is shown in Figure 3-3. The average pose uncertainty ε (3.1b) for each configuration is provided in the legend.

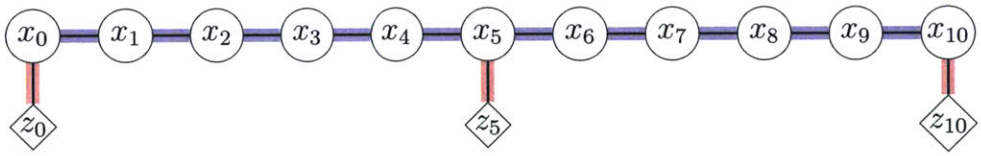


Figure 3-3: The optimal configuration of the pose graph shown in Figure 3-1 of those evaluated in Figure 3-2. The pose position error covariance after optimizing this pose-graph is shown in blue in Figure 3-2.

3.2 Definition of Utility

We define the utility of a location for loop-closure, or *location utility*, as the covariance reduction over *all* poses in the map resulting from all predicted loop-closure measurements that involve that location. Therefore, location utility is composed of the following two factors:

- The probability of a location being visited and recognized for a given map, which is equivalent to the probability of generating a loop-closure measurement.
- The covariance reduction due to loop-closing at that location, which is approximately proportional to the distance traveled since the most recent loop-closure.

These two factors correspond to the total number and spatial distribution of the loop-closure measurements in the map. Without knowledge of the specific vehicle trajectory these factors become independent, as will be described in Sections 3.2.1 and 3.2.2, as long as the location database is not used by the path planner.[1]

Linear combination of the above two utility measures yields the total utility, u of a database location, ℓ, i.e.,

$$\mathrm{u}_\ell = \frac{v_\ell}{v_{max}} + \lambda \frac{d_\ell}{d_{max,\ell}} \tag{3.2}$$

where v_ℓ is the joint probability of a route visiting and recognizing the location (described in Section 3.2.1), normalized by the maximum such probability for the region, v_{max}, and d_ℓ is the minimum distance to another database location or the region boundary (described in Section 3.2.2), normalized by the maximum distance to any database location, $d_{max,\ell}$. λ is a tunable weighting parameter used to control the trade-off between measurement probability and dispersion. We recommend $\lambda = 1$ for general usage, and Section 4.1 will discuss situations in which this parameter might be changed in practice. While this does form the core definition of location utility, this definition is not necessarily complete. For example, it may make sense in some application-specific situations to include an additional term in (3.2) to represent the

[1]Feedback of the location database to the path planner could be accounted for by iterating the location selection process described in the following chapter but has been left for future work.

complexity of the task to be performed at that location relative to position uncertainty (e.g., a street intersection with four streets may more challenging to navigate than one with only three). The following subsections explain the two terms comprising the location utility in detail.

3.2.1 Measurement Probability

Term v_ℓ in (3.2) is defined as the probability that a particular location, ℓ, will contribute a loop-closure measurement to the pose-graph. Thus v_ℓ encodes environment and routing structure and is the joint probability that a vehicle route within the environment passes through and recognizes the location (i.e., successfully performs data association). Specifically, using Bayes' rule, we define v_ℓ as follows:

$$v_\ell = P(\text{visit})P(\text{recognize}|\text{visit}) = P(\text{visit}, \text{recognize}) \tag{3.3}$$

$P(\text{visit})$ is inherently dependent on the trajectory, $\boldsymbol{x} = \{x_{t_0}, x_{t_1}, ...\}$, where x_{t_0} is the vehicle pose at time t_0. When the trajectory is directly observable, $P(\text{visit})$ is readily computed:

$$\text{P}(\text{visit}|\boldsymbol{x}) = \begin{cases} 1 & \ell \in \boldsymbol{x} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

However, the true trajectory is unobservable because the location selection process occurs prior to vehicle navigation. We therefore marginalize out this trajectory dependence by integrating over all possible trajectories, $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\}$:

$$P(\text{visit}) = \int_{\boldsymbol{X}} P(\text{visit}|\boldsymbol{x})P(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \tag{3.5}$$

Where $P(\boldsymbol{x})$ gives the probability of trajectory $\boldsymbol{x}$ occurring.

For the case of a network environment, $P(\text{visit})$ can be computed using a modified form of the betweenness centrality [187] of the graph of traversable routes in the environment. Betweenness centrality measures the extent to which a node in a graph falls on the optimal path between pairs of other nodes, and originated as a measurement of influence in social networks. Thus, having a large betweenness centrality in a transportation network indicates that many routes pass through that location. For purposes of computing $P(\text{visit})$, the standard definition of betweenness centrality is modified to additionally include the route endpoints as "central" locations, as they provide additional opportunities for loop-closure measurements. For all routes from location s to location t, the modified betweenness centrality is computed as the ratio of the number of routes that traverse location ℓ, $n_{st}(\ell)$, divided by the total number of routes in the region, n_{st}:

$$P(\text{visit } \ell) = \sum_{s \neq t} \frac{n_{st}(\ell)}{n_{st}} \tag{3.6}$$

These routes should be computed using the same method as the vehicle's path planner. An example of the probability of location visit in a street network computed using the modified betweenness centrality (3.6) is shown in Figure 3-4.

The recognition probability, $P(\text{recognize}|\text{visit})$, represents the probability that a constraint can actually be generated if the vehicle visits the location and activates its sensors. If sample sensor measurements are available for all locations in the environment, the location recognition probability can be estimated from a saliency score such as [133]. Otherwise, we assume that all locations in the environment are equivalently recognizable: $P(\text{recognize}|\text{visit}) = \text{constant}$.

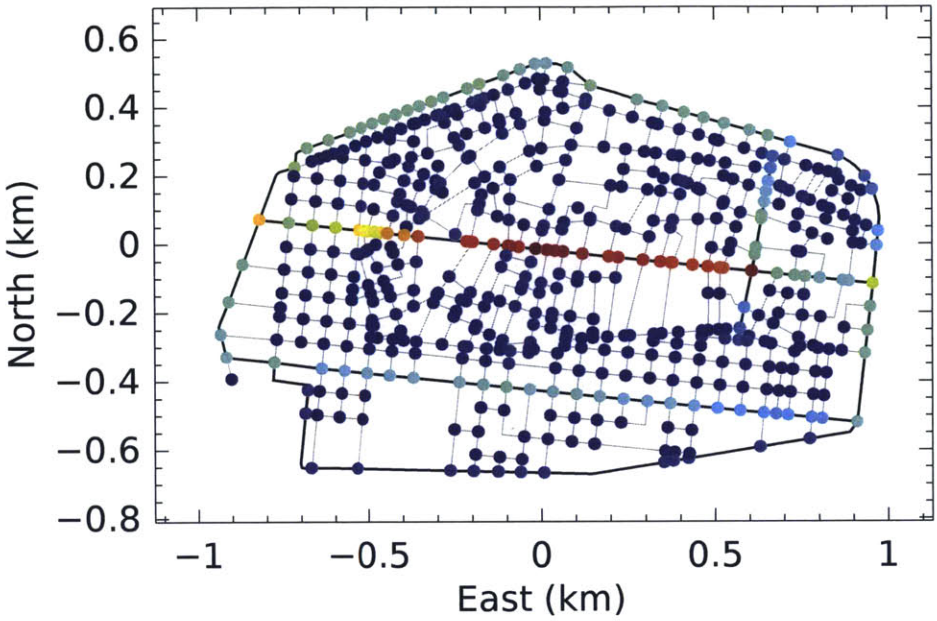


Figure 3-4: Street map of Island of Malé (also see Figure 5-3a), whose intersections are colored to show probability of visit (3.3), ranging from dark red (maximum) to dark blue (minimum).

3.2.2 Location Dispersion

The actual covariance reduction resulting from a loop-closure in a map is difficult to predict, but is approximately proportional to the distance that a vehicle has traveled since the last loop-closure, assuming that the trajectory estimate has achieved steady state (i.e., past poses can be considered well-estimated). This is due to the fact that accumulated odometry error is typically proportional to the distance traveled by the vehicle, similar to propagation error accumulation in a Kalman Filter [23]. In the case of a known trajectory, this distance, d, is easily computed as the distance traveled since the most recent loop-closure location, and maximizing the spatial distribution entails evenly spacing the locations along the trajectory.

We want to achieve this same effect even when the true vehicle trajectory is unknown. Thus, intuitively, we want to maximally distribute the loop-closure sites about the navigable environment, or maximize their *dispersion*. We can estimate the dispersion of the loop-closure measurements by replacing the true distance from the last loop-closure measurement, d, with its lower bound, d_ℓ, which represents the shortest

distance a vehicle could travel from a previous loop-closure before reaching a given location, ℓ. Distance d_ℓ can be computed using either the Euclidean or shortest path distance from location ℓ to either a location contained within the location database, D, (and thus capable of generating a loop-closure constraint) or the region boundary, B:

$$d_\ell = \min \left[\text{dist}(\ell, \ell_D),\ \lambda_B\ \text{dist}(\ell, B) \right] \tag{3.7}$$

where "dist" represents some distance function (e.g., Euclidean distance) and λ_B is used to reduce the impact that the region boundary has on d_ℓ and can be chosen with *a priori* knowledge of how close the vehicle is expected to get to the boundary and how frequently this will occur. Neglecting to include some level of boundary repulsion ($\lambda_B > 0$) tends to highly value locations that lie on or near the region boundary, which may be unlikely for the vehicle to traverse.

The dispersion of a set of locations, h, such as the location database D, is defined as the maximum of the dispersions of the individual locations, d_ℓ:

$$\begin{aligned} h(D) &= \max d_\ell \\ &\ell \in D \end{aligned} \tag{3.8}$$

This is called the "p-dispersion" function [188, 189] or "maximin" dispersion [190, 191]. This definition of the dispersion of a set of locations is convenient for location selection, as maximizing the minimum distance between any two locations in the set maximizes the spatial dispersion of the entire set and serves to strongly discourage clustering. Maximizing the sum of (3.7) for all locations, called the "remote-psuedoforest" or "maxisum" dispersion problem, encourages clustering and is poorly suited for greedy optimization [192], which we will use in Chapter 4 for database selection.

3.3 Extension to Landmark Utility

A landmark-based navigation system seeks to minimize the error of the trajectory estimate using full-rank odometry and low-rank landmark measurements, where landmarks are orientationless points in space with associated appearance descriptors stored in the navigation database. Rather than compensating for odometry drift using full-rank loop-closure measurements, a landmark-based system instead uses line-of-sight observations of landmarks to bound error growth. If we assume that all landmark line-of-sight measurements are equally precise, the two factors influencing the pose error covariance are the number and spatial distribution of the landmark measurements. The average pose covariance for the trajectory is reduced as additional measurements are added, and, for a fixed number of constraints on a particular pose, the average pose covariance is minimized when the constraints are maximally distributed around the sensor field of view.

As a simple motivating example, we consider the case of a single, stationary camera at an altitude of 100 meters pointing directly downward (along nadir). We can compute the pose of the camera if we are given at least three measurements to pre-mapped landmarks by using nonlinear least squares optimization to minimize the reprojection error of the observations in the camera frame, as described in [42]. We simulate a 1024×1024 pixel camera with an approximately 60-degree field of view and assume that all measurements have correct data association to the landmark database and are equally precise. We summarize camera pose uncertainty using the determinant of the position error covariance matrix, which represents the volume of the position uncertainty. In Figures 3-5 and 3-6, we see that pose uncertainty is reduced both as additional measurements are added (Figure 3-5) and as the measurements are increasingly distributed within the image (Figure 3-6). The four-measurement case in Figure 3-5 is identical to the case of 1024-pixel spacing in Figure 3-6.

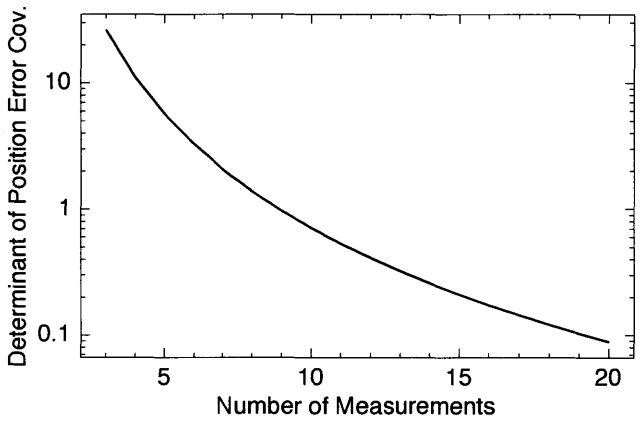


Figure 3-5: Line-of-sight observations of terrain landmarks are simulated in the camera view equally-spaced on the maximum-radius circle centered at the center of projection. The computed camera position uncertainty decreases as additional observations are incorporated into the pose estimation.

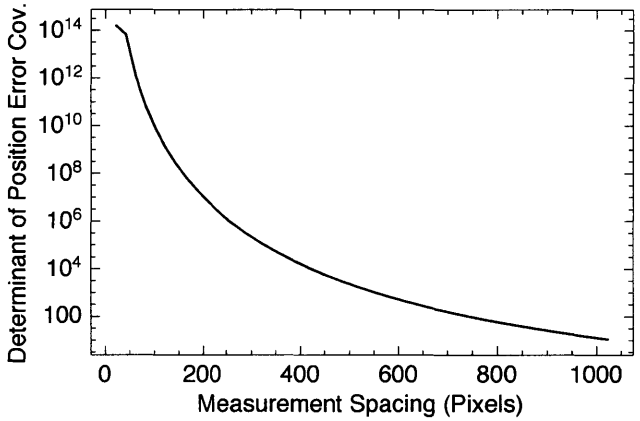


Figure 3-6: Four line-of-sight observations of terrain landmarks are simulated in the camera view in a square centered at the center of projection. The computed camera position uncertainty decreases as the observations (and thus also the landmarks) grow increasingly distant from one another.

For the case of landmark line-of-sight measurements, v_ℓ in (3.2) is defined as the probability that a particular landmark, ℓ, will contribute a measurement to the pose-graph at some vehicle pose, x. Term v_ℓ still encodes environment and routing structure, but is now the joint probability that the vehicle *views* and recognizes a landmark, rather than *visiting* a location in space. This means that a landmark is viewable from many vehicle poses, rather than a maximum of once throughout the course of a point-to-point trajectory, as in (3.4). Similarly to (3.3), v_ℓ is defined using Bayes' rule as:

$$v_\ell = P(\text{view})P(\text{recognize}|\text{view}) = P(\text{view}, \text{recognize}) \tag{3.9}$$

The landmark recognition probability $P(\text{recognize}|\text{view})$, represents the probability that a constraint can actually be generated if the vehicle views the landmark location and activates its sensors. The recognition probability can be estimated from sample sensor measurements, if available. Otherwise we assume that all landmarks in the environment are equivalently recognizable: $P(\text{recognize}|\text{view}) = \text{constant}$.

Because landmarks can be viewed from multiple poses within a trajectory, the probability of viewing a landmark, $P(\text{view})$, must be formulated with respect to individual poses, x, rather than trajectories (vectors of poses), $\boldsymbol{x}$, as in (3.5). This requires the addition of a second integral to the computation of $P(\text{visit})$ in (3.5) to integrate over each trajectory in addition to the space of all trajectories, $\boldsymbol{X}$:

$$P(\text{view}) = \int_{\boldsymbol{X}} \int_{\boldsymbol{x}} P(\text{view}|x)P(x|\boldsymbol{x})P(\boldsymbol{x})\,\mathrm{d}x\,\mathrm{d}\boldsymbol{x} \tag{3.10}$$

where $P(\boldsymbol{x})$ is the probability of a particular trajectory occuring and $P(x|\boldsymbol{x})$ is the probability of pose x occurring within trajectory $\boldsymbol{x}$. $P(\text{view})$ is computed by simulating every possible trajectory and counting the number of times each landmark is observed, weighting by the probability of each trajectory occurring. Once a trajectory has been simulated, the probability of a pose occurring is computed as follows:

$$\mathrm{P}(x|\boldsymbol{x}) = \begin{cases} 1 & x \in \boldsymbol{x} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

In practice, Monte Carlo trajectory simulations can be used to sample $P(\boldsymbol{X})$, as demonstrated in Section 6.4.

The pose error covariance reduction resulting from a landmark measurement is approximately proportional to its 2D-space distance in the sensor field of view, d, to the next-nearest observed landmark. In the example case used earlier of a camera pointing perpendicularly to a planar surface, maximally distributed landmarks in 3D world-space result in maximally distributed observations in 2D image-space. If the camera is pointed off-nadir or the world is non-planar, the maximally distributed observations will no longer exactly correspond to equally-spaced landmarks due to the angular projection. However, these deviations are relatively minor as long as the pointing angle is reasonably small.

The expected distance in the sensor field of view, d_ℓ, can be approximated by the 3D-space Euclidean distance from a specified landmark's location, ℓ, to the nearest landmark in the database, ℓ_D. Despite the true sensor pointing angle being unknown because the landmark database must be selected prior to vehicle navigation, this approximation holds for reasonably small pointing angles because points far apart in space tend to also project far apart in the sensor field of view. Thus we can maximize the spatial distribution of the landmarks in 3D-space as a proxy for maximizing the distribution of their observations in 2D projective-space, again computing d_ℓ as:

$$d_\ell = \min\left[\text{dist}(\ell_i, \ell_D),\ \lambda_B\ \text{dist}(\ell_i, B)\right] \tag{3.12}$$

Similarly to (3.7), we additionally consider the distance from the landmark to the flight region boundary, B, to prevent points on the boundary from being weighted excessively high, as they are less likely to persist in the sensor field of view, and λ_B is again used to down weight the impact of the region boundary. Notice that (3.12) ends up being identical to (3.7), as both methods are simply maximizing the minimum distance between any landmark/location and its nearest neighbor in the database.

3.4 Chapter Summary

The primary goal of localization and SLAM systems is to determine the vehicle's trajectory as accurately as possible. As described in Section 3.1, the accuracy of a vehicle's trajectory estimate can be quantified using the trajectory estimate's uncertainty in the absence of specific measurements. The measurement configuration that minimizes the trajectory estimate's uncertainty has the maximum number of maximally-distributed, maximally-precise measurements.

Section 3.2 defined a metric of map location utility for predicted loop-closure or localization measurements. This utility is a combination of the probability of a measurement occurring at that location and the location's relative spatial dispersion with respect to other locations at which measurements can occur. These two terms become independent if the specific vehicle trajectory is unobservable, enabling location valuation prior to vehicle operation. The measurement probability, defined in Section 3.2.1, is the joint probability of the vehicle visiting and recognizing a location. The location's spatial dispersion, defined in Section 3.2.2, is defined as the minimum distance to any other measurement-enabled location.

Section 3.3 generalized this definition of location utility to accommodate low-rank landmark line-of-sight measurements and unconstrained trajectory spaces. The resulting landmark utility metric is analogous to the location utility metric for purposes of the maximum-utility database selection methods described in Chapter 4.

Chapter 4

Location Database Selection

The previous chapter introduced a metric of location utility, which quantifies the value for vehicle navigation of predicted loop-closure or localization measurements predicted to occur at a particular location, and showed how this metric can be computed for locations in a navigation database. This chapter discusses how to construct such a navigation database to maximize its total utility for a given number of locations.

We first formulate the location selection problem as a nonlinear integer program in Section 4.1, which enables all locations in an environment to be sorted by their relative importances for vehicle navigation. Optimal location selection turns out to be NP-hard, but Section 4.2 describes a greedy approximation algorithm for efficient location selection with time complexity bounded by the product of the number of unselected locations in the region and the database size. In order to further speed up location selection for large regions, we introduce a hierarchical selection approach in Section 4.3 that additionally enables efficient database updates when a portion of the environment changes.

4.1 Location Selection Problem Formulation

We seek to select the maximum-utility set of locations to include in a limited-size navigation database, D. The utility of a set of locations is defined as the sum of expected measurements resulting from all locations in the set (3.3) and the dispersion of the set (3.8). Given all possible locations in a region, L, we can formulate the optimal database selection as a nonlinear integer program (IP) subject to the allowable database size constraint, $|D|$:

$$\begin{aligned} \text{Find} \quad \arg\max_{\boldsymbol{\ell}} \quad & \frac{\boldsymbol{\ell}^T \boldsymbol{v}}{v_{max}} + \lambda \frac{h(L(\boldsymbol{\ell}))}{h_{max}} \\ \text{subject to} \quad & \boldsymbol{\ell}^T \boldsymbol{\ell} = |D| \\ & \ell_i \in \{0,1\} \ \forall i \end{aligned} \tag{4.1}$$

where $\boldsymbol{\ell}$ is a vector of binary switching variables representing all possible database locations ($|\boldsymbol{\ell}| = |L|$), v_{max} and h_{max} are normalizing constants, and λ is a tunable weighting parameter. $h(L(\boldsymbol{\ell}))$ is the p-dispersion function defined for a set (3.8), and $\boldsymbol{v}$ is a vector containing the visit probability of each location in L, as defined in (3.3) or (3.9), which is proportional to the number of measurements predicted to occur at each location. The problem is nonlinear because the dispersion $h(L(\boldsymbol{\ell}))$ is a nonlinear function of $\boldsymbol{\ell}$, as shown in (3.8), and cannot be written in vector form.

Vector $\boldsymbol{v}$ and $h(L(\boldsymbol{\ell}))$ must be normalized to remove the unit conflict between probability and distance, as in (3.2). The normalization constants, v_{max} and h_{max}, are the optimal objective values for their respective single-optimization problems:

$$v_{max} = \arg\max_{\boldsymbol{\ell}_v} \boldsymbol{\ell}_v^T \boldsymbol{v} \tag{4.2a}$$

$$h_{max} = \arg\max_{\boldsymbol{\ell}_h} h(L(\boldsymbol{\ell}_h)) \tag{4.2b}$$

$$\begin{aligned} \text{subject to} \quad & \boldsymbol{\ell}_v^T \boldsymbol{\ell}_v = \boldsymbol{\ell}_h^T \boldsymbol{\ell}_h = |D| \\ & \ell_{v,i} \in \{0,1\}, \ \ell_{h,i} \in \{0,1\} \ \forall i \end{aligned}$$

Determination of the optimal normalization constants is NP-complete (see Section 4.2.2) and dependent on both the environment and database size for this IP formulation, but Section 4.2.1 provides an effective heuristic normalization approach applicable to the greedy approximation algorithm that follows.

λ is a tunable weighting parameter used to control the trade-off between the predicted number and dispersion of the measurements. Specifically, $\lambda < 1$ favors locations expected to be frequently visited and $\lambda > 1$ favors locations that are evenly distributed. In practice, λ should be slightly increased if the $P(\text{visit})$ is especially large in a small number of clustered regions and otherwise generally small (e.g., resulting from a trajectory aiming for a particular goal location), and slightly decreased if $P(\text{visit})$ contains many distributed but minor local maxima (e.g., sinusoidal in nature with a low amplitude). This tradeoff decreases in significance as $|D|$ grows large, and $\lambda = 1$ is sufficient in most cases. Section 5.3.1 further discusses the effects and sensitivity of varying λ.

4.2 A Greedy Algorithm for Location Selection

Directly solving the constrained optimization problem (4.1) is an instance of IP with binary switching variables (also called "0 1 integer programming"), which is NP-hard [158]. The decision version of this problem, in which we are given a location database and must determine whether a database with higher utility exists, is equivalent to one of Karp's famed 21 NP-complete problems [193]. Thus, in order to make location database sorting computationally tractable, we use a greedy approximation algorithm that iteratively adds the remaining location with the highest utility score to the database.

By greedily growing the database, we implicitly assume that the optimal database of size N is included in the optimal database of size $N + 1$. While this assumption does not always hold – its accuracy is discussed in Section 4.2.2 – it does lead to a nice property. The information contained in an accurate loop-closure measurement is

non-negative, i.e., adding an additional constraint to the pose-graph cannot increase the trajectory position uncertainty. We can extend this statement to say that adding a location to the database (without removal of another location) guarantees at worst a null-effect on pose uncertainty, because, for any possible trajectory, adding a location to the database can never result in fewer loop-closure detections. This property means that the utility of a greedily-selected location database is nondecreasing with database size for *any* trajectory. The accuracy of the greedy assumption is discussed in Section 4.2.2.

The greedy database growing problem is formulated as such: Given a list of all possible locations, L, and a database of locations, $D \subsetneq L$, find the optimal next location ℓ:

$$\begin{aligned} \text{Find} \quad & \arg\max_{\ell} \quad \boldsymbol{v}_\ell \; + \; \lambda \, \boldsymbol{d}_\ell \\ \text{subject to} \quad & \ell \in L, \; \ell \notin D \end{aligned} \tag{4.3}$$

where λ is the weighting parameter from (4.1). Vector $\boldsymbol{v}$ provides the probability of visiting each location, computed using Algorithm 1. Vector $\boldsymbol{d}$ provides the distance from each location to its nearest neighbor in D, computed using Algorithm 2. The elements of both $\boldsymbol{v}$ and $\boldsymbol{d}$ are normalized to the range $[0, 1]$ to remove the unit conflict. A more detailed discussion of the normalization approach is provided in the following section. Because dispersion (vector $\boldsymbol{d}$) is undefined for $|D| < 2$, the database is initialized using the location corresponding to the maximum of $\boldsymbol{v}$.

This greedy selection approach is significantly more efficient than the combinatorially-complex IP formulation (4.1), which runs in $\mathcal{O}(\binom{|L|}{|D|})$. Algorithm 1 runs in $\mathcal{O}(|L|)$, Algorithm 2 in $\mathcal{O}((|L| - |D|)|D|)$, and solving (4.3) in $\mathcal{O}(|L| - |D|)$. Thus the full algorithm runs in $\mathcal{O}((|L| - |D|)|D|) < \mathcal{O}(|L||D|)$ for each iteration, and is especially efficient when computing small databases ($|D| << |L|$). Despite location databases typically being computed prior to vehicle navigation, the greedy algorithm is efficient enough to run in on-demand or real-time scenarios, such as would be required for online database management, one of the research directions suggested for future work in Section 7.1.1.

Algorithm 1 Compute Normalized $\boldsymbol{v}$

Inputs:
Locations L,
Location database D,
Visit probabilities $\boldsymbol{v}_L$
$\boldsymbol{v} \leftarrow \text{zeros}(|L|)$
for $x \in L$ **do**
  if $x \notin D$ **then**
    $\boldsymbol{v}[x] \leftarrow \boldsymbol{v}_L[x]$
  end if
end for
$\boldsymbol{v} \leftarrow \boldsymbol{v} / \max \boldsymbol{v}$ ▷ Normalize

Algorithm 2 Compute Normalized $\boldsymbol{d}$

Inputs:
Locations L,
Location database D,
Region boundary B,
Boundary repulsion weight λ_B
$d \leftarrow \text{zeros}(|L|)$
for $x \in L$ **do**
  if $x \notin D$ **then**
    $\boldsymbol{d}_D \leftarrow \text{zeros}(|D| + 1)$
    for $k \in D$ **do**
      $\boldsymbol{d}_D[k] \leftarrow \text{dist}(x, k)$ ▷ Euclidean distance
    end for
    $\boldsymbol{d}_D[|D| + 1] \leftarrow \lambda_B \text{ dist}(x, B)$ ▷ Distance to region boundary B
    $\boldsymbol{d}[x] \leftarrow \min \boldsymbol{d}_D$
  end if
end for
$\boldsymbol{d} \leftarrow \boldsymbol{d} / \max \boldsymbol{d}$ ▷ Normalize

4.2.1 Normalization

The two phenomena forming the core of location utility – the expected number of loop-closure measurements and dispersion – use fundamentally different units, posing a challenge when jointly maximizing them. $P(\text{view})$ is unitless and constrained to lie within the range $[0, 1]$, although, in practice, the maximum viewing probability is often much lower than 1. The database dispersion, $h(D)$, defined in (3.8), is the maximum distance between any location and its nearest neighbor, where both locations are contained in D, and has units of distance. Compounding the challenge, the database dispersion $h(D)$ decreases as the database grows in size and can have a large range of values. For example, $h(D)$ might be on the order of meters when $|D| = |L|$ but on the order of kilometers when $|D| = 2$ during the first greedy selection iterations. The dispersion value will typically vary by orders of magnitude during database selection, which makes it difficult to set the tuning parameter, λ, such that it provides a consistent effect throughout the selection process and is independent of the database size.

The underlying goal of database selection is to achieve a balance between the expected number of measurements and their dispersion, and to use λ to manually tune this desired balance. To ensure that the effect of λ is applied consistently during database selection and is independent of the desired database size, we re-normalize the two objectives on every iteration prior to selecting the optimal location. The final step in Algorithms 1 and 2 is dividing vectors $\boldsymbol{v}$ and $\boldsymbol{d}$ by their respective maximum values, constraining their values to the range $[0, 1]$. The greedy algorithm then selects the location, ℓ, that maximizes the sum of $\boldsymbol{v}[\ell] + \lambda \boldsymbol{d}[\ell]$. For the standard case of $\lambda = 1$, the per-location utility during each selection iteration is now constrained to the range $0 \leq \boldsymbol{v}[\ell] + \lambda \boldsymbol{d}[\ell] \leq 2$. Increasing λ to 1.1 results in a 10% bias towards measurement dispersion during location selection, for example.

This normalization approach provides several benefits. Most notably, the values of both competing objectives are normalized to the same range, allowing direct combination without a unit conflict. The algorithm selects the location with the best trade-off between the two competing objectives at each iteration, independently of the database size or specific environment properties. The weighting parameter, λ, intuitively and consistently weights the objectives independently of their units, the database size, and properties of the environment, and seamlessly handles the wide range of magnitudes through which the dispersion decreases as the database grows. Furthermore, this normalization approach does not interfere with the selection process. It is easy to show that greedily maximizing either objective independently would result in the same set of locations with or without normalization, as the normalization approach does not change the order in which the locations are selected. This detail is a subtle requirement of the proof of the approximation accuracy of the greedy selection approach provided in the following section.

4.2.2 Approximation Accuracy

The greedy location selection approach (4.3) provides a computationally tractable but suboptimal approximation to the NP-hard optimal location database selection problem (4.1). This section provides an analysis of the approximation accuracy of greedy location selection.

The utility objective function is a summation of two independent objectives. The first objective, v, represents the probability of visiting and recognizing a location and corresponds directly to the number of expected loop-closure measurements resulting from including the location in the navigation database (see Section 3.2.1). The second objective, d, represents that location's relative spatial dispersion with respect to the locations contained in the database (see Section 3.2.2).

Taken independently, maximizing only the number of expected loop-closure measurements, v, is both submodular and greedy-optimal. Submodular objective functions are particularly well-suited for greedy optimization because every future greedy selection is guaranteed to increase the value of the objective equally or less than the previous step (the diminishing returns principle) [158]. This is intuitive, as selecting the location with the maximum weight on each iteration will clearly give the collection of locations with maximum total weight. The normalization scheme discussed in the previous section and used in Algorithm 1 does not alter the selection order. This objective function is sometimes called a "maxisum" objective [191], as it maximizes the sum of the expected loop-closures.

The performance of greedily maximizing the dispersion objective, d, is more complex. Recall from Section 3.2.2 that dispersion is defined as the maximum of the minimum distances from each location (or landmark) in the database to any other location in the database (3.8) and thus we are maximizing the maximum of these minimum distances (sometimes called a "maximin" objective). This objective is classified as a p-dispersion problem, which "choose p out of n given points such that the minimum distance between any pair of chosen points is as large as possible" [188, 190, 191], and its decision formulation is NP-complete (i.e., determining whether a subset of points exists with greater dispersion than some other subset) [189]. This objective is known in the field of operations research as the "Obnoxious Facility Placement Problem," where it is applied for purposes such as minimizing the market overlap between a chain of restaurants [192] or locating undesirable facilities, such as garbage depots or nuclear reactors [191].

The obnoxious facility placement problem is commonly solved using a greedy approach, and multiple authors have shown that the worst-case performance of the standard greedy selection approach is twice the optimum, or *2-approximate* [194, 188, 195]. Erkut, et. al [190] showed experimentally that greedy selection typically finds a solution within 80%-90% of the true maximum, especially when n is large ($|L|$). In continuous space, this greedy heuristic is equivalent to the "largest empty sphere"

problem in computational geometry [196, 197]. The normalization scheme discussed in the previous section and used in Algorithm 2 does not affect the order in which locations are selected by the greedy heuristic. The analysis that follows neglects the boundary repulsion term, λ_B, described in Section 3.2.2 (effectively $\lambda_B = \infty$), which effectively shrinks the region boundaries.

Summing the two objectives comprising location utility results in a maximin-maxisum bi-objective optimization problem. This problem is sometimes called the "semi-obnoxious facility placement problem" in the field of operations research, where the motivation is generally minimizing facility cost while maximizing dispersion or coverage [198]. Related formulations can present maximally valuable but minimally redundant search results to users of online search engines or automatically summarize user comments on online news articles [194].

Dasgupta, Kumar, and Ravi [194] show that greedy maximization of a p-dispersion objective and a submodular objective (such as maximizing the expected loop-closure measurements) results in, *at worst*, $1/4^{\text{th}}$ of the optimal value. The following proof follows similar arguments to show that the greedy location selection approach (4.3) results in a database that guarantees *better than* $1/4^{\text{th}}$ the utility of the optimal location database.

Theorem 1. *For $|D| > 1$, there is a polynomial time algorithm that obtains a location database with better than a $\frac{1}{4}$-approximation to the maximum utility database.*

Proof. Recall from (3.2) that database utility $u(D)$ is a combination of two independent functions of the locations in the database, D: the probability of visiting the locations, $v(D)$, and location dispersion, $h(D)$.

$$u(D) = v(D) + \lambda h(D) \tag{4.4a}$$

$$v(D) = \Sigma_{i \in D} P(\text{visit}_i) \tag{4.4b}$$

$$h(D) = \min_{\{\alpha,\beta\} \in S} \text{dist}(\alpha, \beta) \tag{4.4c}$$

Where S is the set of unordered pairs $\{\alpha, \beta\}$ where $\alpha \in D$ and $\beta \in D$.

We begin by running the greedy selection algorithm twice: first with v as given and $h \equiv 0$ and second with $v \equiv 0$ and h as given, resulting in solution databases D_v and D_h, respectively. Let O_v and O_h be the optimal databases in each case, and let O be the optimal database.

The visit probability objective, $v(D)$, is known to be greedy optimal:

$$v(D_v) = v(O_v) \tag{4.5a}$$

$$u(D_v) \geq v(D_v) \tag{4.5b}$$

The utility of database D_v must be greater than or equal to $v(D_v)$ alone because $h(D_v) \geq 0$ by definition.

As established independently by [194, 188, 195], the greedy maximization of the p-dispersion objective is 2-approximate:

$$h(D_h) \geq \frac{1}{2} h(O_h) \tag{4.6a}$$

$$u(D_h) \geq \lambda h(D_h) \tag{4.6b}$$

The utility of database D_h must be greater than or equal to purely the dispersion of the database, $h(D_h)$, because $v(D_h) \geq 0$ by definition.

Using an averaging argument, we see that if $u(D)$ is the larger of $u(D_v)$ and $u(D_h)$, then it must be greater than their average:

$$D = \underset{X \in \{D_v, D_h\}}{\arg\max}\ u(X) \tag{4.7a}$$

$$u(D) \geq \frac{u(D_h) + u(D_v)}{2} \tag{4.7b}$$

A bi-objective optimization cannot result in a larger value of one of its objective functions than the corresponding single-objective optimization for that objective:

$$v(O_v) \geq v(O) \tag{4.8a}$$

$$h(O_h) \geq h(O) \tag{4.8b}$$

The sum of the two single-objective results is thus greater than or equal to the bi-objective result:

$$v(O_v) + \lambda h(O_h) \geq v(O) + \lambda h(O) = u(O) \tag{4.9}$$

Combining (4.5b) and (4.6b) with (4.9):

$$u(D_v) \geq v(O_v) \tag{4.10a}$$

$$u(D_h) \geq \frac{1}{2}\lambda h(O_h) \tag{4.10b}$$

$$u(D_v) + u(D_h) \geq v(O_v) + \frac{1}{2}\lambda h(O_h) \tag{4.10c}$$

We now get a lower bound on the approximation accuracy by combining (4.7b), (4.9), and (4.10c):

$$\begin{aligned} u(D) &\geq \frac{1}{2}(u(D_v) + u(D_h)) \\ &\geq \frac{1}{2}(v(O_v) + \frac{1}{2}\lambda h(O_h)) \\ &> \frac{1}{4}(v(O_v) + \lambda h(O_h)) \\ &> \frac{1}{4}u(O) \end{aligned} \tag{4.11}$$

□

This proof establishes that for any set of locations, the better result of greedily maximizing the number of loop closures or greedily maximizing the dispersion will give better than one fourth the database utility of the true maximum. While this does provide a helpful theoretical bound on the result, the greedy selection algorithm of Section 4.2 is expected to perform much better than this bound, in general, for two

reasons. First – as the results in Chapters 5 and 6, and especially Figure 5-8, will show – the normalized greedy selection approach out-performed either of the single-objective results in all cases tested. Recall that the proof states that the better of these two single-objective cases is guaranteed to have greater than 25% of the optimal utility. Second, if in (4.6b) we replace the hard theoretical accuracy of 50% for the p-dispersion problem with the 84% experimental lower bound computed by Erkut, et. al [190] when selecting either 20 or 60 points from a set of 200 (and with increasing accuracy as the number of candidate points in their study grew), the lower bound increases to 42% accurate for the utility-based greedy location selection approach. If a guaranteed error bound is required in practice, the greedy selection should be additionally run for the two single-objective cases, and the selected database with the maximum utility should be used, as this database will be better than a $\frac{1}{4}$-approximation of the optimal database.

4.2.3 Multiple Greedy Step Location Selection

A standard greedy optimization algorithm selects the most optimal next location at each iteration, given the current location set. However, greedy algorithms can be reformulated to select the best location given the current selection set *and* the predicted future selections that would follow. In general, the greedy algorithm will achieve a better approximation of the true optimum when this "look-ahead" range is increased, and increasing the look-ahead range to the full size of the desired database is equivalent to performing a brute-force search for the global optimum if a fully recursive search is performed. For the class of functions that can be described as submodular, meaning every future selection is guaranteed to increase the value of the objective function by less than the previous step (the diminishing returns principle), the accuracy of greedily optimizing the objective provably increases with the number of look-ahead steps [158]. While the location selection problem is not submodular, Section 5.3.2 shows experimentally that increasing the look-ahead range still achieves a better approximation of the true optimum utility database for a network environment.

We can get an indication of the experimental approximation accuracy of the algorithm by comparing the result of the standard greedy database selection problem to that of a multi-step variant. If the selection results are similar, this indicates that the single-step greedy algorithm provides an efficient approximation of the true maximum utility location database. Algorithm 3 provides a non-recursive multi-step implementation of the greedy selection algorithm from (4.3), and Figure 5-11 shows a comparison of the navigation performance achieved by single- and multi-step greedily-selected databases for a vehicle operating in a street network. This implementation does not use full recursion due to the associated computational load, which would be equivalent to a brute force search, and instead makes the greedy-optimal selection at each future iteration. That is, for each candidate location, ℓ, we compute the next $K-1$ locations that *would* be selected in future selection steps, D_{future}, and then compute the utility of ℓ for the database $D_{multi} = D \bigcup D_{future}$. In the single-step algorithm ($K = 1$), $D_{future} = \emptyset$. The normalization approach presented in Section 4.2.1 is still valid for the multi-step greedy selection algorithm.

The recursive greedy algorithm presented in Algorithm 3 consists of three nested loops. The outermost loop controls the size of the database, adding one location to the database, D, on every iteration. The second loop, beginning on line 5, computes the utility of all locations $\ell \in L$ given the current database. This loop begins by forming a temporary database, D_{tmp}, containing all locations in D and the candidate location ℓ. The third loop, beginning on line 8, then greedily adds $K-1$ more locations to D_{tmp}. Lines 13 and 14 remove location ℓ from D_{tmp} and then compute vectors $\boldsymbol{v}$ and $\boldsymbol{d}$. ℓ must first be removed from D_{tmp} to avoid having its values in $\boldsymbol{v}$ and $\boldsymbol{d}$ set to zero. Line 15 then computes the utility of location ℓ and stores it in vector $\boldsymbol{u}$. Once the second loop has completed and the utility of each candidate location has been computed, the location with the maximum utility is added to D in line 18. This process repeats until the database reaches the desired size, N.

Algorithm 3 Multi-Step Greedy Database Selection

Inputs:
Locations L,
Database size N,
Greedy steps K,
Visit probabilities $\boldsymbol{v}_L$,
Region boundary B,
Weighting factor λ,
Boundary weighting factor λ_B

$D \leftarrow \arg\max_i \boldsymbol{v}_L[i]$ ▷ Initialize selection set
while $|D| < N$ **do**
  $\boldsymbol{u} \leftarrow \text{zeros}(|L|)$

  for $\ell \in L$ **do**
    if $\ell \notin D$ **then**
      $D_{tmp} \leftarrow D \bigcup \ell$ ▷ Add ℓ to temporary database D_{tmp}

      while $|D_{tmp}| < (|D| + K)$ AND $|D_{tmp}| < N$ **do**
        $\boldsymbol{v} \leftarrow \text{getWeightedV}(\boldsymbol{v}_L, L, D_{tmp})$ ▷ See Algorithm 1
        $\boldsymbol{d} \leftarrow \text{getWeightedD}(L, D_{tmp}, B, \lambda_B)$ ▷ See Algorithm 2
        $D_{tmp} \leftarrow D_{tmp} \bigcup \arg\max_i (\boldsymbol{v}[i] + \lambda\, \boldsymbol{d}[i])$
      end while

      $\boldsymbol{v} \leftarrow \text{getWeightedV}(\boldsymbol{v}_L, L, D_{tmp} \setminus \ell)$ ▷ Exclude ℓ from D_{tmp}
      $\boldsymbol{d} \leftarrow \text{getWeightedD}(L, D_{tmp} \setminus \ell, B, \lambda_B)$
      $\boldsymbol{u}[\ell] \leftarrow \boldsymbol{v}[\ell] + \lambda\, \boldsymbol{d}[\ell]$ ▷ Utility of location ℓ
    end if
  end for

  $D \leftarrow D \bigcup \arg\max_i \boldsymbol{u}[i]$ ▷ Add maximum utility location to database
end while

Outputs:
Location database D

4.3 Hierarchical Selection using Map Tiles

While the greedy selection method described above is highly efficient, it requires that $P(\text{visit})$ be computed prior to selection, such as by using the modified betweenness centrality (3.6) in the case of a network environment. Unfortunately, determination of $P(\text{visit})$ is both computationally expensive and requires knowledge of the entire operational environment and trajectory space. For the case of navigating in a street network, this involves running Dijkstra's algorithm to completion, starting at every location in the map, which is $\mathcal{O}(n^3)$ for n map locations. While this can be sped up using random sampling to estimate $P(\text{visit})$, this computation eventually becomes prohibitively expensive as the environment grows especially large in size (i.e., the number of candidate locations). The process must also be repeated any time the network is modified, such as due to road construction.

To overcome these limitations, we can divide the map up into a set of subregions, or *tiles*, and select locations hierarchically. This is similar to how online mapping services transmit a set of pre-rendered map tiles to users to reduce latency and bandwidth and then combine the tiles locally on the user devices [199, 200, 201]. We first compute the location dispersion distances and $P(\text{visit})$ locally from only the portion of the region lying within the tile's boundaries. The accuracy of this local approximation of $P(\text{visit})$ is discussed in Section 4.3.1. Greedy location selection is then performed locally within the tile, generating a list of the best locations in the subregion. We next gather the best locations from each tile to form a reduced set of candidate locations and perform greedy location selection on this reduced set using the tile-computed $P(\text{visit})$ values and recomputing the dispersion values. This process can be repeated for as many layers as necessary.

The time complexity of greedy location selection (4.3) is $\mathcal{O}(|L||D|)$ for each iteration, which is the complexity of the dispersion term. Thus location selection with a large set of candidate locations is efficient for the case of selecting a small location database, D, or by reducing the number of candidate locations, L. To speed up the selection

process, we must reduce the time complexity of (4.3), which is achieved by reducing the selection database size for each tile, D_T, which in turn reduces the number of candidate locations for greedy selection one level up in the hierarchy. We use the observation that locally valuable locations tend to also be globally valuable to reduce the selection database size for each tile, computing only the best m locations in each tile, where m is determined according to:

$$m > \frac{N}{T} \tag{4.12}$$

where T is the total number of map tiles and N is the desired location database size at the next hierarchical level. The result is a reduction of the candidate database size for the next hierarchical level from $|L|$ to mT, where $mT << |L|$ if $|D| << |L|$. Boundary effects in the tile-based location selection result in an underestimation of $P(\text{visit})$ on tile edges due to the inability of candidate trajectories to cross tile boundaries. Therefore, tiles should be constructed with overlapping boundaries to help mitigate these tile boundary effects. We further recommend selecting at least three more locations per tile than the minimum given in (4.12) in order to account for tile boundary effects and differences in the dispersion when computed at different hierarchical levels.

4.3.1 Approximating $P(\text{visit})$

The difficulty with computing map database locations using map tiles is that the high-level global structure is no longer fully captured. For example, a fastest-time route confined to a maximum distance of 1 km is unlikely to involve taking a limited-access freeway, though these are typically the roads with the highest traffic. What we need is an approximation for $P(\text{visit})$ that can be computed using only local structure. In the case of street network environments, which can be represented as a graph and are discussed further in Chapter 5, we can coarsely approximate the betweenness centrality (3.6) of a node using Opsahl's weighted popularity measure [202].

Weighted popularity is an expansion of degree centrality, c_d, which counts the number of edges connected to a node of a graph and is a purely local computation with linear complexity with respect to the number of graph nodes. In comparison to degree centrality, node popularity counts only the number of edges *entering* the node in a directed graph. A related graph measure, node strength, w, instead sums the weights of the edges connected to a given node. A popularity strength can similarly be computed by summing the weights of only the edges *entering* that node. Opsahl defines weighted degree centrality, $c_{d,w}$, as a balance of both the degree centrality and strength, controlled by a weighting parameter, α:

$$c_{d,w} = c_d\left(\frac{w}{c_d}\right)^{\alpha} \tag{4.13}$$

Weighted popularity follows from this definition, instead only including the edges entering the node, and represents a pseudo-likelihood that the node will be visited by a shortest path through the graph.

In approximating the betweenness centrality using the weighted popularity for street network environments, we are implicitly applying the assumptions that the street network is efficiently designed and individual street speed limits are proportional to their traffic throughput. This means that we assume a street with a fast speed limit is traversed by more possible vehicle trajectories in an environment than a street with a slow speed limit, and that city designers placed the fastest streets where they are more useful. These assumptions are, of course, never completely true in practice, but they do provide the relationship between local and global structure – one goal of efficient transportation network design is to align the weighted betweenness centrality and weighted popularity of nodes in the network – and thus valuable insight for hierarchical location selection.

4.4 Chapter Summary

This chapter uses the location utility metric defined in Chapter 3 to construct optimal location or landmark databases for vehicle navigation. Section 4.1 formulated maximum-utility location selection as a nonlinear integer program. Unfortunately, maximum-utility location selection is NP-hard and therefore suffers from combinatorial complexity, making the problem computationally intractable. Therefore, we introduced a greedy approximation algorithm for computationally efficient maximum-utility location selection in Section 4.2, which includes a novel normalization approach (described in Section 4.2.1) to accommodate unit conflicts and user preferences, if desired. Section 4.2.2 proves that this greedy approximation algorithm is theoretically better than 25% accurate and better than 42% accurate when considering typical performance of greedily maximizing location dispersion as reported in the literature. The simulation results presented in Chapter 5 indicate that typical performance of the greedy algorithm considerably exceeds these minimum-accuracy bounds.

Despite its relative efficiency, greedy location selection grows computationally expensive for especially large sets of candidate locations. Section 4.3 introduces a hierarchical location selection approach that enables greedy location selection on layers. The vehicle's operational environment (i.e., the region encompassing all candidate locations) is first divided into subregions. Locations are selected locally for each subregion, and the best locations from each subregion are then combined to form the set of candidate locations at the next hierarchical level. Because these subregions might not succeed at locally capturing global-level structure, Section 4.3.1 describes how the measurement probability at each location can be locally approximated for network environments using the weighted degree centrality metric.

Chapter 5

Street Map Navigation

In this chapter, we demonstrate the benefit to pose-graph SLAM when using limited-size, maximum-utility navigation databases selected using the greedy algorithm presented in Chapter 4. An optimal location database is inherently trajectory-specific – a truly optimal database would contain only locations traversed by the trajectory – however, the navigation database must be generated prior to navigation when the true trajectory is unknown. We must therefore evaluate our database reduction strategy for many potential trajectories within an environment. Unfortunately, no existing localization dataset has these characteristics, so we developed a new, open-source simulator capable of generating relative-pose measurements for multiple trajectories through a network of traversable paths, which is described in Section 5.1.

In Section 5.2, we perform maximum-utility database selection in a variety of city environments by sorting locations in the map by their predicted value for loop-closure. We then use the vehicle simulator to generate pose-graphs of long-duration trajectories using these maximum-utility navigation databases in Section 5.3 and compare the trajectory estimation accuracy when navigating with databases selected using various alternative location selection approaches. We extend this comparison in Section 5.4 to hierarchically-selected location databases for a large city. Much of the results presented in this chapter were previously published in [203].

5.1 Street Map Simulator

We built a brand-new routing and pose-graph simulator in `Julia` [204] that can simulate a taxi-like vehicle driving in a city along an optimal route through a succession of waypoints and use the vehicle trajectory to generate a pose-graph from simulated odometry and loop-closure measurements. This simulator is divided between two open-source code packages developed using the `Julia` [204] programming language. The `OpenStreetMap.jl` package[1] provides mapping and routing functionality, and is described in Section 5.1.1. The `PoseGraphSimulation.jl` package[2] provides functionality for generating random routes through the environment and then simulating, solving, and evaluating the associated pose-graphs, and is described in Section 5.1.2. Section 5.1.3 describes the evaluation metrics provided by `PoseGraphSimulation.jl`, which are used to quantify the accuracy of trajectory estimations and compare the utility of navigation databases.

`OpenStreetMap.jl` uses publicly-available map data from `OpenStreetMap.org` to represent the street network for a given region as a directed graph weighted by either edge distance or travel time. The simulator generates routes through the network by randomly selecting 50 waypoints within the network and then computing the shortest-distance or fastest-time driving route (respecting one-way streets) through them using Dijkstra's algorithm (described in Section 2.5.1). For fastest-time routing, we assign reasonable speed-limits to the six most significant classes of streets that are provided by `OpenStreetMap.org` (residential, freeway, etc.), ignoring paths corresponding to driveways and parking lots. For simplicity and without loss of generality, we assume that the streets are infinitesimally narrow.

The simulator establishes poses at every street intersection and every 20 meters along the route, and generates simulated relative-pose odometry measurements between all successive poses. The simulator generates a loop-closure measurement whenever the vehicle traverses a location described by the navigation database. We limit the nav-

[1] Available at https://github.com/tedsteiner/OpenStreetMap.jl.git.
[2] Available at https://github.com/tedsteiner/PoseGraphSimulation.jl.git.

igation database to contain only locations corresponding to street intersections in order to limit the computation required for greedy location selection. Loop-closure measurements are generated to only the pose associated with the first time the vehicle traversed that location. Therefore, if the vehicle has not yet traveled through the intersection, a measurement cannot be generated. We assume that loop-closure detection is not directionally dependent, that loop-closures provide full-rank measurements, and that loop-closures have guaranteed detection. These assumptions are not requirements of the approach but serve to simplify and clarify the results. These limitations are easily accounted for in practice by adding additional candidate locations with specific heading angles or explicitly modeling the location detection probability, as described in Section 3.2.1. The case of low-rank measurements is discussed in Chapter 6.

In order to facilitate direct comparison of location selection methods, the simulator generates a route, odometry measurements, and *all* possible loop-closure measurements (including additive Gaussian noise) for *any* possible navigation database. We can then create multiple pose-graphs for a single trajectory with identical measurement noise, adding loop-closure constraints according to any desired location database (set of map intersections). This way these pose-graphs all contain identical measurement noise for every measurement they have in common, allowing their solutions to be directly compared. The iSAM [67] and RISE [65] algorithms are then used to assemble and solve the pose-graph SLAM problem. Due to the time required to simulate and solve thousands of pose-graphs (Figure 5-7 represents 10,000 pose-graphs, for example), the results presented were typically computed in batch mode, falling back to iSAM's incremental mode whenever the batch solver failed to find a solution due to poor initial guess values.

5.1.1 OpenStreetMap.jl

The `OpenStreetMap.jl` Julia package was developed to facilitate producing driving simulations with realistic street routing and speed limits using data from OpenStreetMap, which is an open, user-contributed repository of global map data.[3] This map data is free to use for any purpose and can be viewed online at `OpenStreetMap.org`. OpenStreetMap data can be downloaded in a variety of data formats, including the `OSM` XML format.

`OpenStreetMap.jl` has grown in scope to become a general-purpose library providing tools for researchers working with OpenStreetMap data and is available from the official Julia package repository. This section provides an overview of how the package is used to produce a driving simulation for the purposes of this thesis. However, the package provides a variety of additional tools, including detailed map display, which are detailed in the official package documentation online.[4] The results in this thesis were generated using Version 0.8 of the package.

`OpenStreetMap.jl` enables convenient parsing of OSM data files, as well as map display, cropping, routing, and filtering. The standard process used to load an OSM datafile and prepare for vehicle routing in Julia is as follows:

1. Load the OpenStreetMap.jl package into the global Julia namespace:

   ```
   using OpenStreetMap
   ```

   This loads all OpenStreetMap.jl types and functions into the namespace.

2. Parse the OSM XML file:

   ```
   nodesLLA, highways, buildings, features = getOSMData( filename )
   ```

   Where `filename` is the path to the `OSM` XML datafile. `nodesLLA` is a dictionary of node ID numbers indexing their locations in latitude, longitude, altitude (LLA) coordinates, and `highways` is a dictionary of highway ID numbers indexing to highway data. The `buildings` and `features` variables are unused.

[3] All map data presented in this work is copyright OpenStreetMap contributors and available under the Open Database License.

[4] Available online at http://openstreetmapjl.readthedocs.org.

3. Convert the map to East, North, Up (ENU) Cartesian coordinates:

   ```
   nodes = OpenStreetMap.ENU( nodesLLA, lla_reference )
   ```

   Where `lla_reference` is the point in LLA to set as the origin of the ENU frame.

4. Crop the map data to specified boundaries:

   ```
   cropMap!( nodes, bounds, highways=highways )
   ```

   Remove all map data outside a specified boundary region, `bounds`, which provides the limits of the vehicle's operational environment. When a highway crosses the boundary, a new node is interpolated to lie on the boundary and added to `nodes` and the corresponding highway object.

5. Extract the highway classes and filter out all non-road highways:[5]

   ```
   roads = roadways( highways )
   ```

   ```
   highways, roads = roadsOnly( highways, roads, classes )
   ```

   Where `classes` is a list of street classifications, according to the following:

   1. Motorway: limited-access divided highway (Interstate Highways in U.S.)
   2. Trunk: inter-regional arterial divided highway
   3. Primary: major routes not classified as "motorway" or "trunk"
   4. Secondary: typically one lane each direction
   5. Tertiary: minor roads for local traffic
   6. Unclassified: narrow, paved public highways
   6. Residential: minor roads in residential areas
   7. Service: driveways, parking lots, etc.
   8. Pedestrian street: pedestrians have right of way

   For results presented in this thesis, routable highways are limited to classes 1-6.

6. Find all highway intersections:

   ```
   inters = OpenStreetMap.findIntersections( highways )
   ```

   Database locations are selected only from highway intersections for the purposes of this thesis.

[5]Within the OpenStreetMap community, all paths are referred to as "highways," including sidewalks and cycleways.

7. Consolidate nearby highway intersections:

```
clusters = findHighwaySets( highways )
clust_map = findIntersectionClusters( nodes, inters, clusters, max_dist=15 )
replaceHighwayNodes!( highways, clust_map )
intersections = findIntersections( highways )
```

Intersecting divided highways can result in four or more intersections due to the handling of oneway streets. This process consolidates intersections within 15 meters of identically named streets for the purpose of location selection, as they will otherwise artificially deflate each other's $P(\text{visit})$.

8. Create a street network object:

```
segments = segmentHighways( nodes, highways, intersections, roads )
network = createGraph( segments, intersections )
```

To simplify routing tasks, `OpenStreetMap.jl` precomputes the directed graph representation of the street network and encapsulates it in `network`, which contains all data required to compute routes within the environment.

`OpenStreetMap.jl` additionally includes functions for computing routes between specified node IDs within the street network. However, for the purposes of generating the long routes used to simulate the pose-graphs in this thesis, these functions have been encapsulated in the separate `PoseGraphSimulation.jl` Julia package.

5.1.2 PoseGraphSimulation.jl

We developed a new Julia package, `PoseGraphSimulation.jl`, to quickly and easily generate pose-graphs using OpenStreetMap data. `PoseGraphSimulation.jl` can additionally write a pose-graph to an iSAM-readable text file, read iSAM's output covariance and solution files into Julia, plot the pose-graph solution (i.e., trajectory estimate), and compute error statistics.

The process for generating a pose-graph with additive Gaussian random noise using the `OpenStreetMap.jl` data objects described in the previous section is as follows:

1. Load the PoseGraphSimulation.jl package into the global Julia namespace:

   ```
   using PoseGraphSimulation
   ```

   This loads all PoseGraphSimulation.jl types and functions into the namespace.

2. Randomly select a set of routing waypoints:

   ```
   waypoints = getWaypoints( network, candidates, min_dist, num )
   ```

   This function recursively grows a list of `num` waypoints, ensuring that a route exists between consecutive waypoints. `network` is the street network object computed in Section 5.1.1, `candidates` is the list of intersection IDs (node IDs at intersections) that can be selected as waypoints, and `min_dist` is the minimum Euclidean distance between consecutive waypoints.

3. Compute the route between consecutive waypoints:

   ```
   route_major, route_metric = computeRoute( network, waypoints, fastest )
   ```

   `route_major` is an ordered list of intersections traversed by the route, and `route_metric` is either the route time or distance. Parameter `fastest` determines whether the minimum time or distance route is computed. When using fastest routes, the road classes (stored within the `network` object) are used to apply average speed limits to individual road segments. Datasets generated for this thesis used `OpenStreetMap.jl`'s default speeds, `OpenStreetMap.SPEED_ROADS_URBAN`, as OpenStreetMap does not provide speed limits for all highways.

4. Interpolate the route:

   ```
   route = interpolateRoute!( nodes, route_major, pose_spacing )
   ```

   Add additional `nodes` along `route_major` every `pose_spacing` meters.

5. Generate truth poses:

   `poses = getPoses2D( nodes, route )`

   Produces an ordered list of poses at each node along `route`. Poses are composed of `x` (East), `y` (North), and θ (heading) values.

6. Generate odometry measurements:

   `odom_factors = getPoseOdometry( route, poses, odom_cov )`

   Returns a list of objects containing all data required to generate the associated odometry factors in iSAM, with Gaussian noise added according to the covariance matrix, `odom_cov`.

7. Generate loop-closure measurements:

   `loop_factors = getLoopFactors( route, poses, loop_locs, loop_cov )`

   Returns a list of objects containing all data required to generate the associated loop-closure factors in iSAM, with Gaussian noise added according to the covariance matrix, `loop_cov`. For every node ID in `route` that is also in the unordered set `loop_locs` (the list of locations in the navigation database, at which loop-closures can occur), the function generates a new loop-closure measurement from the associated pose in `poses` to the pose associated with the *first* occurrence of that node ID in the route. No measurement is generated if the specified pose is the first traversal of the location in `route`.

8. Write truth data to text file:

   `writeTruthFile( fname, poses, anchors=loop_locs, waypoints=W, flags=F )`

   Write truth poses to text file `fname`. Optional arguments can additionally write the list of loop-closure node IDs, `loop_locs`, waypoint node IDs, `W`, and a `String`-indexed dictionary of parameter flags, `F`, but these values are not used for pose-graph solution evaluation.

9. Write pose-graph measurements to iSAM-readable text file:

   `makePoseGraph( loc_sets, route, poses, odom_factors, loop_factors, fnames )`

   Composes the pose-graphs and generates N iSAM-readable text files, where N is the length of the input vectors `loc_sets` and `fnames`. This allows common measurements to generate multiple pose-graphs to compare the effects of varying the available loop-closure sites, which are stored in `loc_sets`.

The pose-graph can now be solved using iSAM's command line interface directly from Julia. In some cases, iSAM can get "stuck" while trying to solve the pose-graph, so the process can be called with an automatic timeout, as shown in Listing 5.1.

```
isam = (`$isam_path -P -U $fname_cov -W $fname_result $fname_graph`)
proc = spawn( isam )
timedwait( ()->process_exited( proc ), isam_timeout )
if process_running( proc )
  kill( proc )
  warn( "iSAM killed (timed out)." )
end
```

Listing 5.1: Robust iSAM call with time limit

In Listing 5.1, if the solution exceeds `isam_timeout` seconds, it can be automatically killed. Option `P` solves the pose-graph using the Powell's Dogleg method, which is more robust than the default Gauss-Newton method, especially when the quality of the initialization point is unknown.

Once the iSAM solver has finished running we can compute the trajectory position estimation error and position error covariance. The process for computing these metrics is as follows:

1. Compute the pose-graph position error:

   `position_rmse = posegraphError( fname_truth, fname_result )`

   The true and estimated poses are parsed from `fname_truth` and `fname_result`. We first use the Kabsch algorithm [205, 206] to compute the relative transform

between the estimated and true pose positions and align them to eliminate the effects of gauge freedom. We then subtract the x and y positions of the true and estimated poses and compute the root mean squared error (RMSE) of these position errors, `position_rmse`.

2. Compute the average estimation error variance of position for the trajectory:

   ```
   covariances = readiSAMOutputCov( fname_cov )
   epsilon = epsilonTrajectory( covariances )
   ```

   Parse the block covariances for each pose from `fname_cov` and compute ε (3.1b), the average position estimation error uncertainty for each pose in the trajectory.

5.1.3 Quantifying Reduction Error

We use the average position error covariance for all poses in the trajectory (3.1) as a performance metric to compare the trajectory estimation accuracy resulting from navigating with various location databases. We refer to this metric as "mean trajectory uncertainty," ε. However, the accuracy of the trajectory estimate when using a limited location database is inherently trajectory-dependent. Therefore, we simulate several vehicle trajectories (typically either 20 or 50, depending on the dataset) and use each of these trajectories to generate a pose-graph for each navigation database presented. Recall that the only difference between pose-graphs simulated for a specified trajectory is the presence or absence of loop-closure measurements at specific locations in space, corresponding to the locations contained in the navigation database. Every pose-graph generated for a given trajectory has equivalent noise for every odometry and loop-closure measurement that they have in common.

We solve each pose-graph corresponding to a reduced navigation database, $D_{reduced}$, using iSAM and determine its corresponding mean trajectory uncertainty, $\varepsilon_{reduced}$. However, this metric is trajectory-specific and therefore cannot be aggregated with values of $\varepsilon_{reduced}$ computed for different trajectories through the same street map, which is highly desirable in order to remove trajectory-specific effects from quanti-

tative database comparisons. Therefore, in order to make meaningful comparisons of pose-graph estimates between different trajectories and street maps, we normalize each pose-graph's mean trajectory uncertainty by the mean trajectory uncertainty achieved when using the "full" location database, ε_{full}, which contains all possible loop-closure locations in the vehicle's operational environment:

$$\varepsilon_{ratio} = \frac{\varepsilon_{reduced} - \varepsilon_{full}}{\varepsilon_{full}} \tag{5.1}$$

We call this metric the "trajectory uncertainty ratio," ε_{ratio}. A small ε_{ratio} indicates that a reduced navigation database provides sufficient loop-closure measurements to achieve a mean trajectory uncertainty approaching that of the full database.

In the figures that follow in this chapter, we plot the average ε_{ratio} for many randomly-generated routes through a street network using a common navigation database to eliminate trajectory-specific effects. These plots show ε_{ratio} averaged over many trajectories for greedily-selected navigation databases increasing in size ($|D|$ in (4.3)). A good location selection strategy results in ε_{ratio} decreasing as quickly as possible as the database size increases, reaching $\varepsilon_{ratio} = 0$ when $D_{reduced} = D_{full} = L$.

5.2 Navigation Database Selection

In this section, we generate maximum-utility navigation databases using the greedy location selection algorithm presented in Section 4.2. Because the street map is a continuous space, we must first discretize the environment into a finite set of candidate locations. For the case of street-network environments, we limit our set of candidate locations for selection to street intersections. Figure 5-1 depicts the distribution of P(visit) for a region in Cambridge, Massachusetts, computed using the modified betweenness centrality at each street intersection. Recall from Section 3.2.1 that locations with large P(visit) correspond to locations expected to result in the most loop-closure measurements in a long-duration vehicle trajectory. Figure 5-2 depicts

the 5, 10, and 15 best loop-closure locations for a vehicle driving within this region, computed using the greedy location sorting/selection algorithm. The greedy algorithm achieves a balance between selecting locations with high $P(\text{visit})$ and locations distant from others already contained in the database and the region boundary.

We additionally performed greedy location selection at a larger scale for five cities with various regular and irregular street networks. Figures 5-3 and 5-4 show the street maps and top 15 highest-utility locations for these cities, as well as all street intersections in each map color-coded by their greedily-computed utility scores. The Island of Malé (Figures 5-3a and 5-3b) is especially interesting because, unlike all other maps tested, it does not have any streets crossing the region boundary and therefore does not display any boundary effects (the region boundary, B, is set well-outside the figure boundaries). Seattle (Figures 5-3c and 5-3d) and San Francisco (Figures 5-4a and 5-4b) both have grid-like street networks, which tend to result in well-dispersed location selections due to $P(\text{visit})$ being consistently large along arterial roads. Lower Manhattan (Figures 5-3e and 5-3f) has a very irregular street layout consisting mostly of one-way streets, which results in the selected locations tending to cluster around bottlenecks and major cross-streets. Moscow (Figures 5-4c and 5-4d) has a radial structure centered on the Kremlin, and we see that locations selected for Moscow tend to lie along either of the two major rings. As Figures 5-3 and 5-4 show, the greedy location selection algorithm is applicable in a wide variety of city topologies, and favors both major intersections and locations spatially distributed between them for all cities tested. In all of the cities, residential areas tend to have uniformly low visit probability distributions due to a lack of through-traffic, resulting in the speckled-blue pattern that is especially apparent in San Francisco (Figure 5-4b).

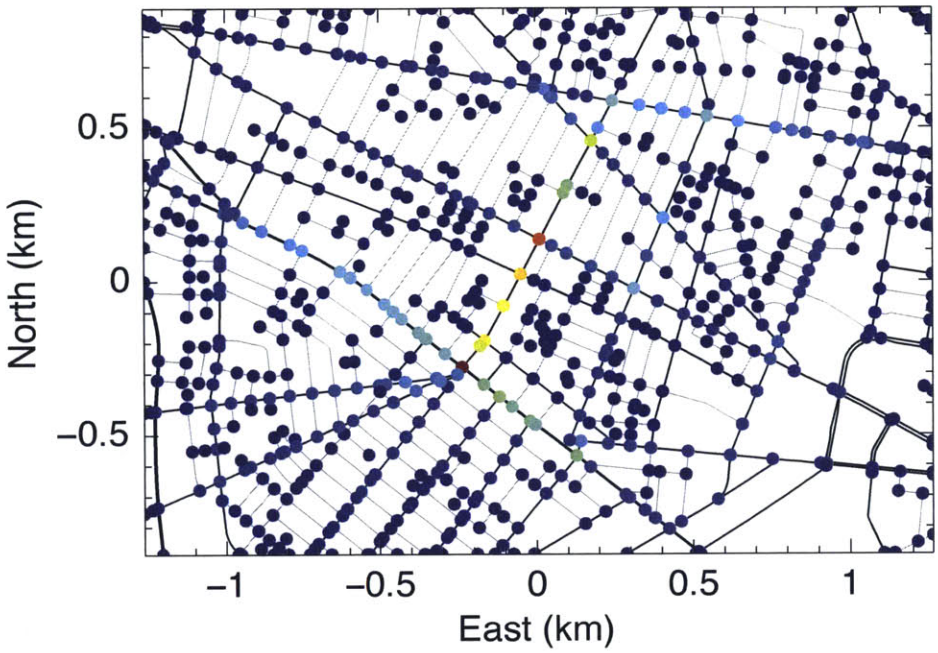


Figure 5-1: Normalized visit probability distribution, $P(\text{visit})$, for a region of Cambridge, Massachusetts, computed using the modified betweenness centrality (3.6). Locations most likely to be visited by a fastest-time point-to-point route through the environment are shown in dark red.

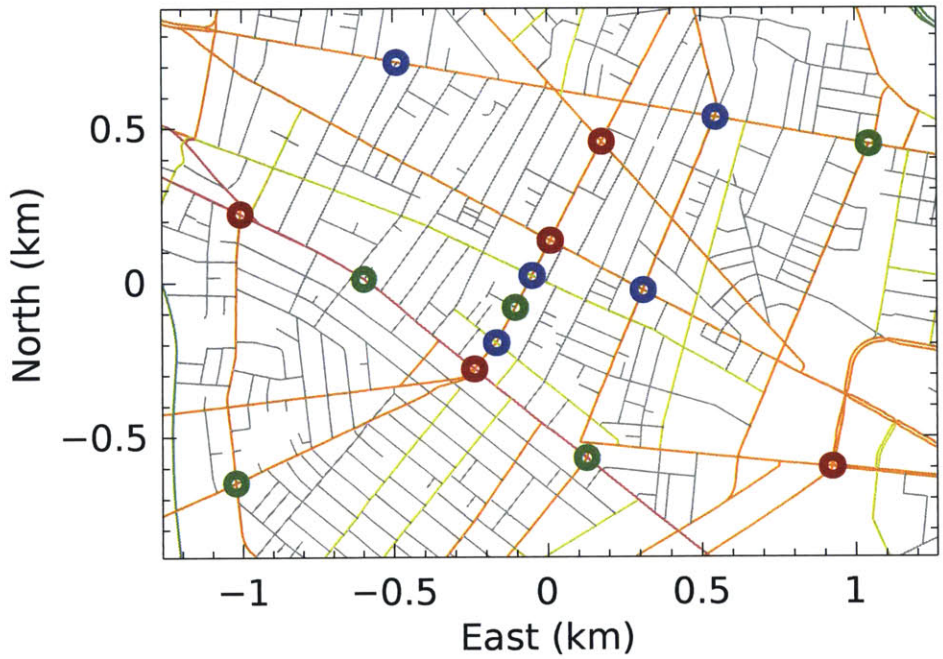
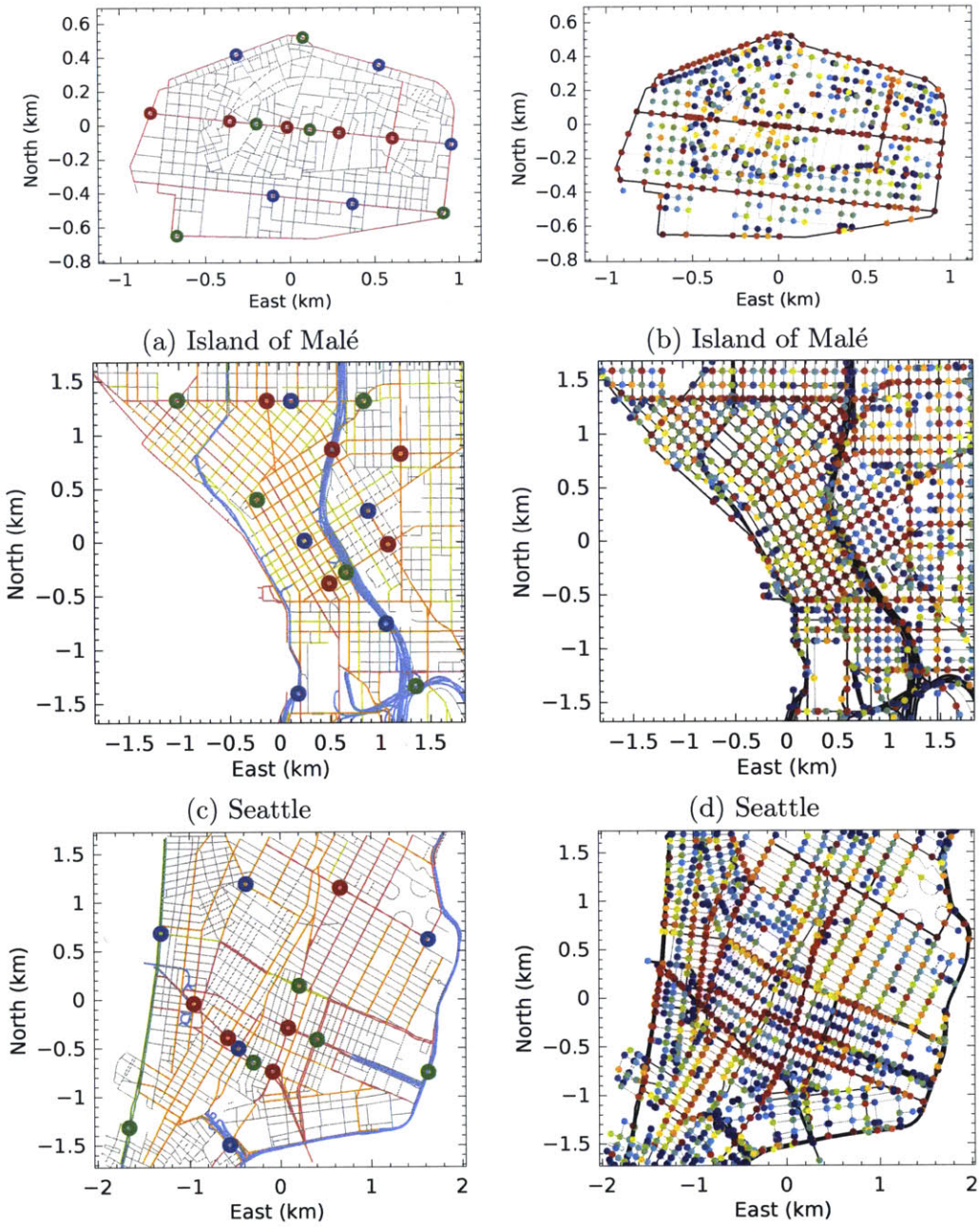


Figure 5-2: Best 5 (red), 10 (red + green), and 15 (red + green + blue) locations selected for a minimal database for a region of Cambridge, Massachusetts. The greedy location selection algorithm achieves a balance between locations with a high probability of being visited, as depicted in Figure 5-1, and locations that are spatially distributed in the environment.



(a) Island of Malé

(b) Island of Malé

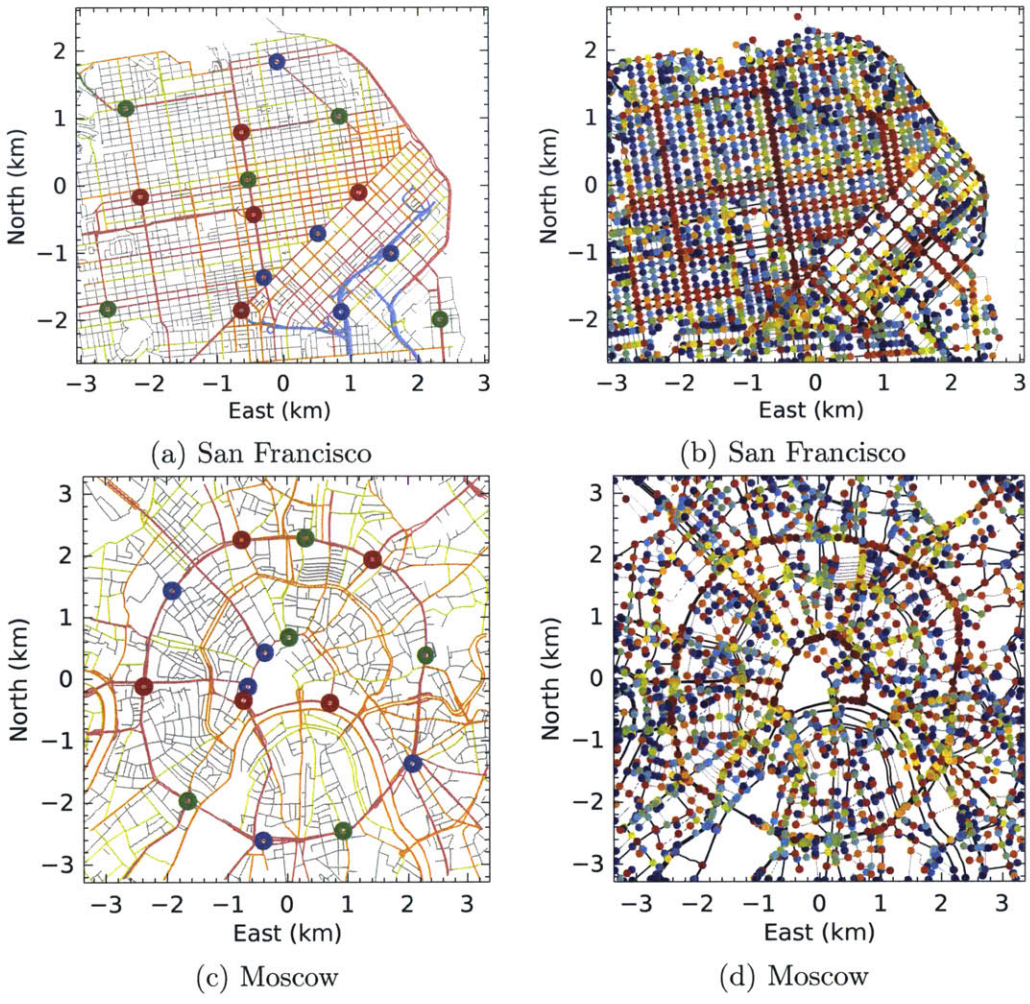
(c) Seattle

(d) Seattle

(e) Lower Manhattan

(f) Lower Manhattan

Figure 5-3: The best 1-5 (red), 6-10 (green), and 11-15 (blue) street intersections for five major cities (left), and all street intersections sorted based on loop-closure utility (right). The maximum database sizes for the maps range from 535 (Malé) to 2,811 (Moscow) locations, increasing from top to bottom. In all cases, our algorithm favors both locations on major highways and locations spatially distributed between them. Continued in Figure 5-4.



(a) San Francisco

(b) San Francisco

(c) Moscow

(d) Moscow

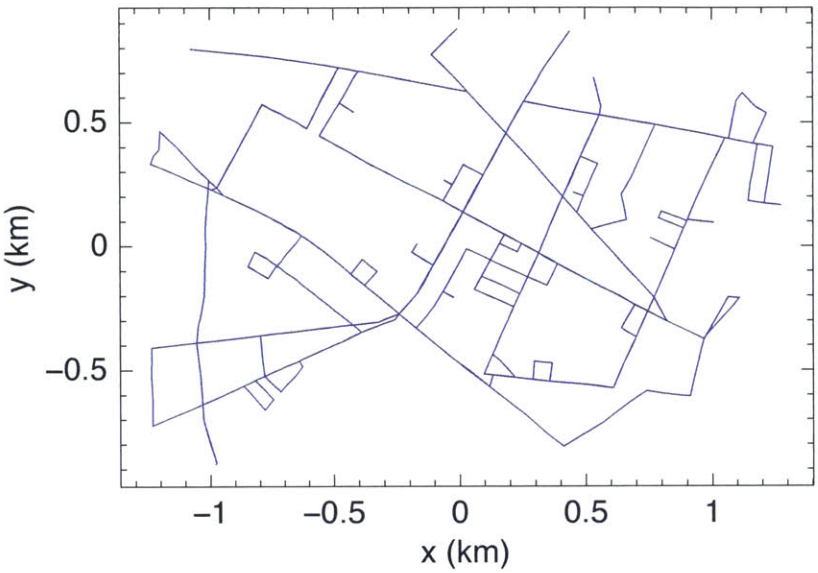
Figure 5-4: Continuation of Figure 5-3.

5.3 Localization Using Reduced Database

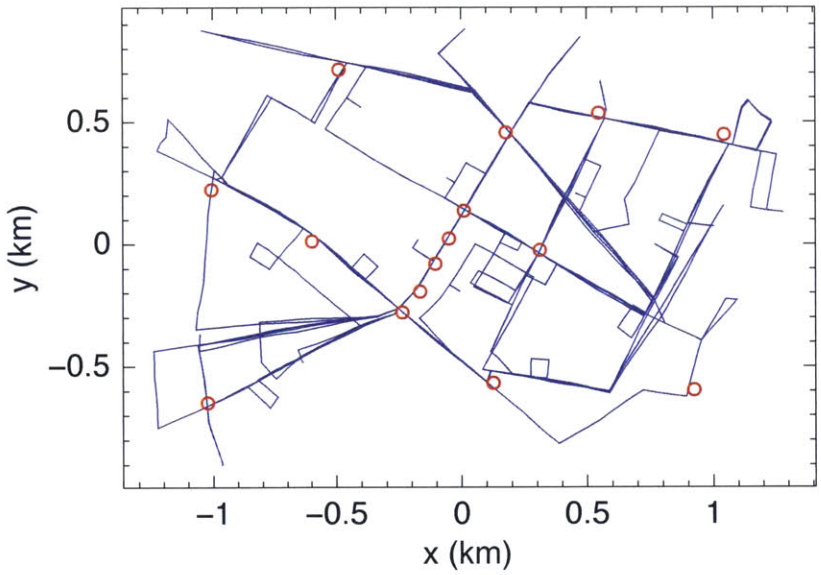
The accuracy of vehicle trajectory estimation using reduced-size navigation databases is dependent on the selection model used to construct the database. We demonstrate this by comparing vehicle trajectory estimates computed from pose-graphs resulting from navigating with various reduced navigation databases. We begin by showing a comparison of trajectory estimates using full and reduced navigation databases. Section 5.3.1 then uses the trajectory uncertainty ratio metric introduced in Section 5.1.3 to compare trajectory estimates computed using navigation databases selected with several selection techniques, demonstrating the value of the maximum-utility greedy location selection approach. Section 5.3.2 compares trajectory estimates using maximum-utility navigation databases selected using single- and multi-step greedy approaches.

Figure 5-5 shows two example pose-graph solutions for a 76.6 km trajectory through Cambridge, Massachusetts, both containing 3,375 poses. Figure 5-5a is computed using the full location database of all 712 street intersections (roughly equivalent to 20% of the poses, though many locations are never actually encountered during the trajectory), and Figure 5-5b is computed with the reduced location database containing the 15 highest-utility locations computed in the previous section and shown in Figure 5-2 (less than 0.5% of all poses in the trajectory).

While the smaller database gives a noisier result, the underlying network structure is still distinctly visible, and the vehicle state knowledge is likely sufficient to carry out basic navigational tasks if the vehicle additionally has local obstacle detection capability and access to the street map for high-level routing in unexplored areas of the map. In addition, by inferring path crossings as intersections, uncertainty-cognizant, pose-graph-based path planners, such as [179, 180, 182], could be used to plan routes within such a reduced pose-graph even without access to the street map.



(a)



(b)

Figure 5-5: Results of solving a simulated pose-graph representing driving around Cambridge, Massachusetts for approximately 75 km using: a) a location database containing all 712 street intersections, and b) a dramatically reduced database of only 15 pre-selected intersections (0.5% of the full database of 3,375 poses). The true positions of the selected database entries are shown in red, and the pose-graph solution was rotated and translated to get the best fit to these true positions in post-processing as it was not natively computed in absolute coordinates. Due to the high utility of the 15 selected locations, the topology of the map is well-captured and the graph is approximately metrically correct, as the 15 selected intersections are very close to their true positions.

5.3.1 Comparison to Alternative Selection Methods

We now compare our location selection approach to four alternative methods: (i) random selection, (ii) maximizing the number of loop-closures, (iii) maximizing the spatial dispersion, and (iv) maximizing the weighted degree centrality.

Random Selection

Figure 5-6 compares the average trajectory uncertainty ratios, ε_{ratio}, resulting from navigating 50 routes with maximum-utility and randomly selected databases containing up to 100 locations in Cambridge, Massachusetts (see map in Figure 5-2). The maximum-utility greedy selection approach results in significantly better accuracy than randomly selecting locations for the navigation database.

Figure 5-7 extends this comparison to the five additional cities whose maps are shown in Figures 5-3 and 5-4. The maximum-utility databases constructed using the greedy selection algorithm outperform their randomly-selected counterparts for all cases tested. The maximum utility databases were all selected using $\lambda = 1$ in (4.3), resulting in an even balance between maximizing the number and dispersion of loop-closure measurements in the pose-graph.

Alternative Objective Function Weighting

Recall that we define location utility (3.2) as the sum of two terms: the probability of generating a loop-closure at that location and its dispersion from other locations in the database. We maximize the utility of a navigation database by jointly maximizing these two terms in the form of a bi-objective optimization (4.1), where the first objective maximizes the number of predicted loop-closures and the second maximizes their dispersion. The trade-off between these two objectives can be controlled using the tuning parameter, λ.

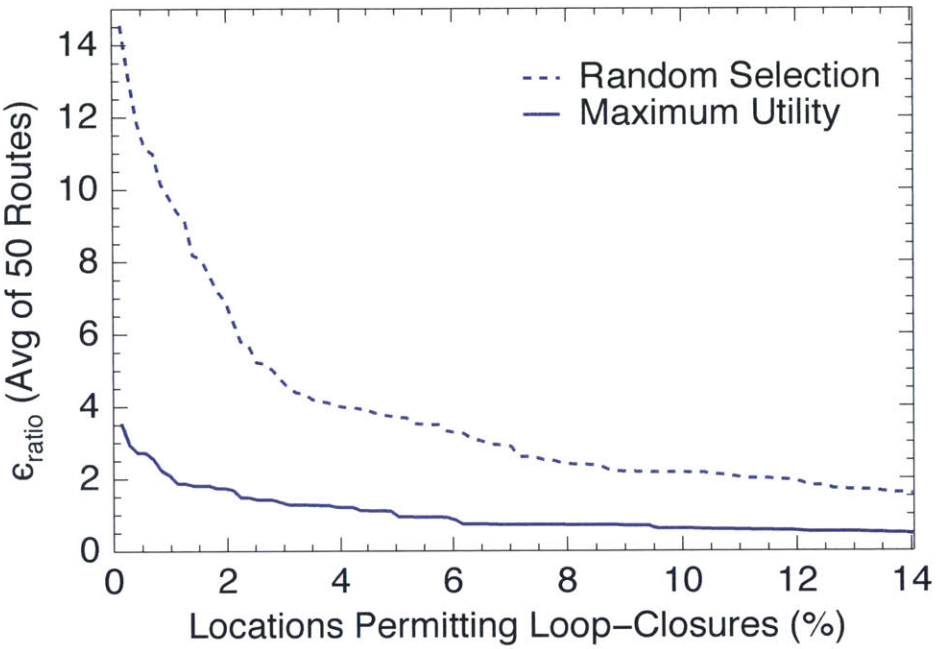


Figure 5-6: Comparison of trajectory uncertainty ratios (5.1), ε_{ratio}, resulting from using maximum-utility and randomly-selected location databases for a vehicle navigating in Cambridge, Massachusetts as the database size increases. Navigating with a complete database containing all 712 intersections results in $\varepsilon_{\text{ratio}} = 0$, and a smaller ε_{ratio} indicates that the database is closer to this ideal.

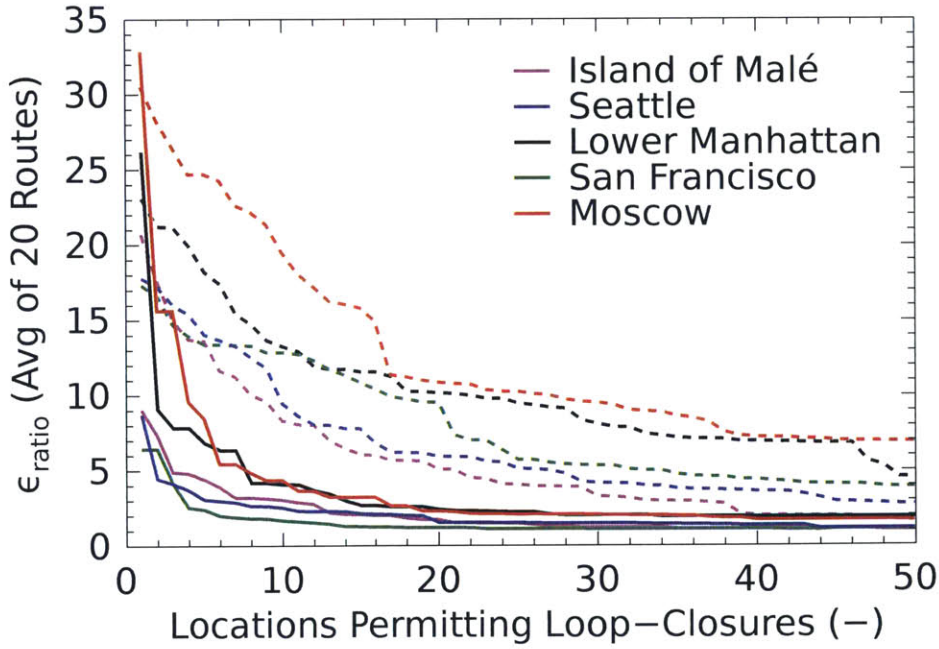


Figure 5-7: Comparison of trajectory uncertainty ratios when using locations selected with the maximum utility approach (solid lines) vs. randomly (dashed lines) for a vehicle navigating in five cities.

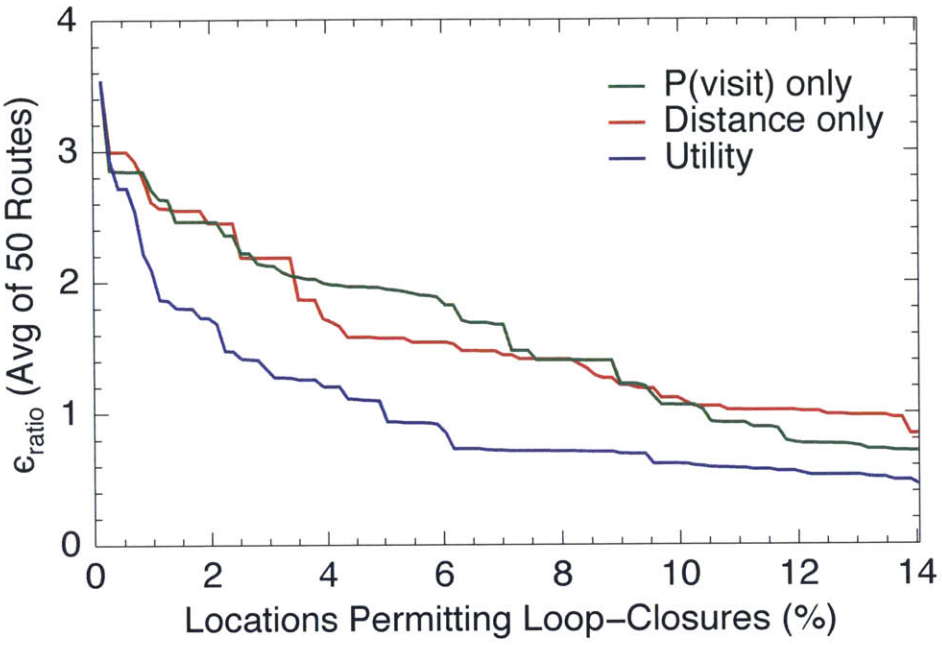


Figure 5-8: Comparison of the effects of the two objectives that form the "location utility" on the trajectory uncertainty ratio. The green line determines location utility using only measurement probability (v), the red line uses only dispersion (d), and the blue line uses an equal combination of the two terms. All three databases are initialized using the location with maximum P(visit) for $|D| = 1$.

Figure 5-8 shows a comparison of the trajectory uncertainty ratios achieved when navigating a vehicle during long-duration trajectories in Cambridge, Massachusetts with databases selected using different values of λ. The three cases in the figure correspond to (i) maximizing the predicted number of loop-closures ($\lambda = 0$), (ii) maximizing the dispersion of predicted loop-closures ($\lambda \gg 1$), and (iii) the recommended even balance between the two ($\lambda = 1$). Both of these objectives are independently valuable, but our balanced combination outperforms both approaches for all database sizes tested (1 to 100 of 725 total locations). Section 4.1 provides guidance on selecting λ according to properties of the measurement probability distribution for the vehicle's operational environment. For street maps in general, we found the results to be fairly insensitive to the specific value of λ chosen, as long as $\lambda \approx 1$. Figure 5-9 shows that varying λ in the range of $0.8 \leq \lambda \leq 1.2$ for the Cambridge street map has very little effect on the resulting trajectory uncertainty.

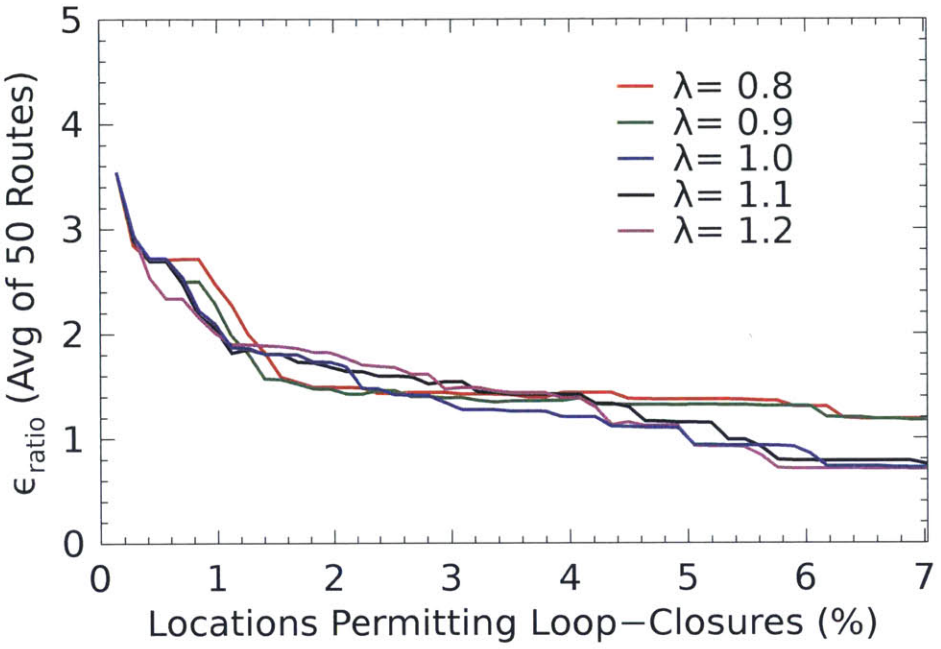


Figure 5-9: The trajectory uncertainty ratio, ε_{ratio}, is generally insensitive to the specific value of the manual tuning parameter, λ, as long as $\lambda \approx 1$.

Recall from Section 4.2.2 that the better-performing database of these two alternative objectives is guaranteed to have greater than $^1/_4{}^{\text{th}}$ the total utility of the optimal location database for the region. The fact that the maximum-utility database considerably outperformed optimizing either objective function independently gives strong evidence that the greedy selection approach exceeds the $^1/_4{}^{\text{th}}$-approximation bound for this region, and similar performance was observed for other regions tested. However, a firm statement of optimality cannot be drawn, as the ε_{ratio} metric is an indirect measure of database utility.

Degree Centrality

Directly evaluating $P(\text{visit})$ for all locations in the environment prior to location selection is computationally expensive. Section 4.3.1 suggested the use of weighted degree centrality (4.13) as an efficient approximation of the betweenness centrality used to compute $P(\text{visit})$, which represents the cumulative strength of all paths *entering* each intersection (i.e., weighted popularity). In this section, we evaluate the accuracy of this approximation.

Figure 5-10 compares the trajectory uncertainty ratio achieved when navigating a vehicle through Moscow (the largest map tested, containing $2,811$ total intersections) with maximum-utility database locations selected using $P(\text{visit})$ computed with the modified betweenness centrality versus estimating $P(\text{visit})$ using weighted degree centrality. The figure additionally compares against databases selected by purely maximizing the weighted popularity centrality of the location database and random selection. It is clear from the figure that none of the alternatives perform as well as the maximum utility database computed with the true $P(\text{visit})$ (computed using the modified betweenness centrality). However, the next-best result, especially after about 25 locations have been selected (0.9% of all candidate intersections), is the utility computed using weighted degree centrality (popularity), which converges to only about twice the $\varepsilon_{\text{ratio}}$ of the betweenness-computed utility. Thus the computationally efficient degree centrality can serve as a viable alternative to betweenness centrality if the betweenness centrality cannot be directly computed, such as in the case of hierarchical location selection on tiled maps, as described in Section 4.3. Results for hierarchical selection using the weighted degree centrality are presented in Section 5.4.

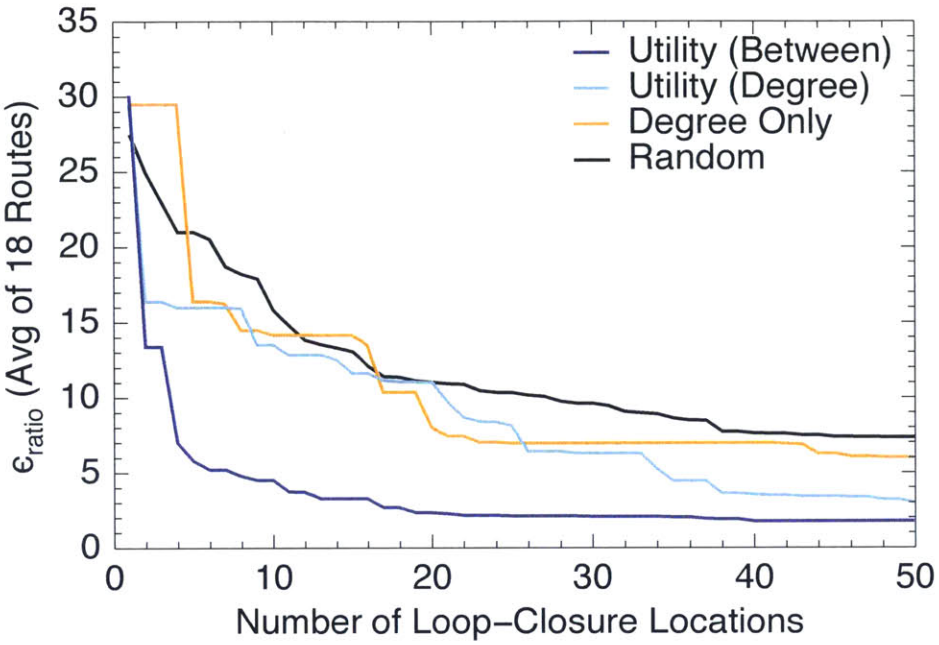


Figure 5-10: Comparison of average pose uncertainties when navigating through Moscow using loop-closure locations selected using the modified betweenness centrality (standard) or weighted degree centrality (efficient) to compute $P(\text{visit})$. While the standard betweenness centrality approach results in the best location database, weighted degree centrality provides a good approximation for larger databases.

5.3.2 Multi-Step Greedy Approaches

The standard greedy location selection algorithm selects the most optimal next location at each iteration. However, as discussed in Section 4.2.3, it is common practice to run greedy optimization algorithms for multiple future iterations in an attempt to select the location with the best "future" value (i.e., when the location database is larger). In general, the greedy algorithm will achieve a better approximation of the true optimum when this look-ahead range is increased. Increasing the look-ahead range to the full size of the desired database is equivalent to performing a brute-force search for the global optimum.

Figure 5-11 shows a comparison of computing the utility using both one and five look-ahead steps on each iteration. The figure shows that the 5-step database almost globally outperforms the 1-step database, as one would expect. Recall that the the evaluation metric, $\varepsilon_{\text{ratio}}$, is an indirect approximation for the actual utility (the

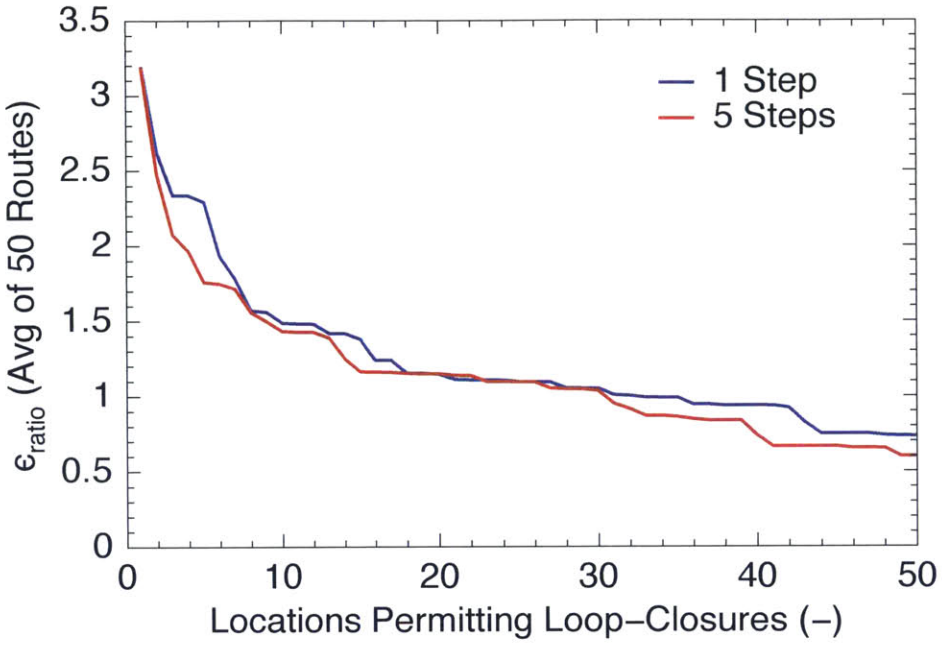


Figure 5-11: Comparison of average pose uncertainties when navigating through Cambridge, Massachusetts using locations selected by the standard 1-step greedy algorithm and a 5-step implementation. The similarity of the results indicates that the greedy selection approach provides a good heuristic for maximum-utility database selection.

absolute value of which is meaningless), and thus can provide only an indication of the optimality of the greedy utility-maximization algorithm. Nonetheless, the 1-step and 5-step solutions achieve very similar error values, which indicates that even the 1-step greedy algorithm is capturing the vast majority of the optimality of the 5-step algorithm and thus serves as a good selection heuristic.

5.4 Tile-based Hierarchical Location Selection

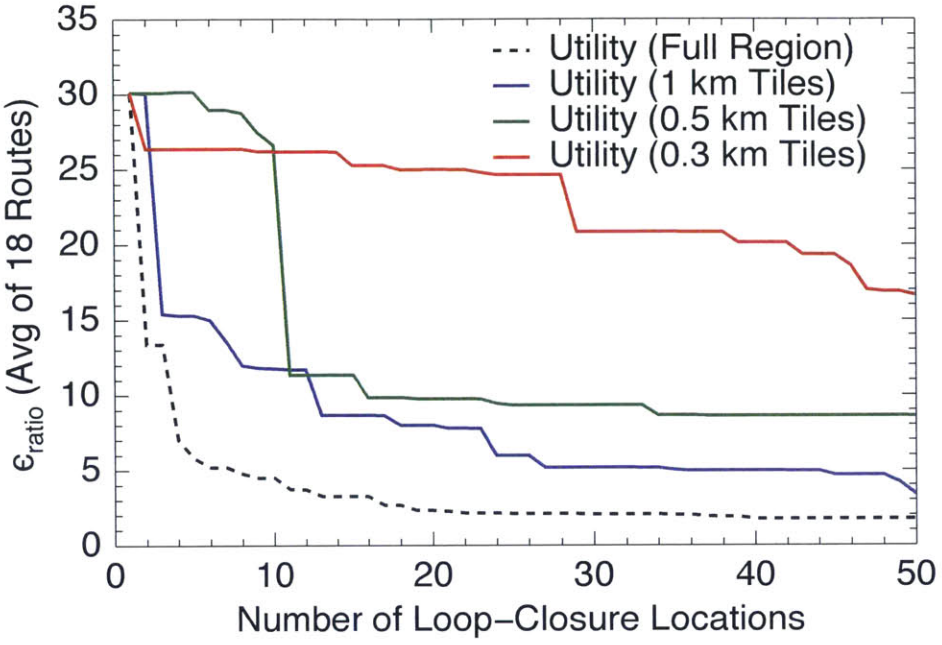
Section 4.3 introduced a hierarchical method for computationally efficient location selection for large regions that divided the region into "tiles." The best locations are first determined for each tile, and these locations are then used to select the best locations for the entire region. This approach could further extend over multiple layers of tiles, if needed.

Figure 5-12a shows the average $\varepsilon_{\text{ratio}}$ for 18 routes through Moscow (the largest map tested) computed using various sizes of square tiles with 25% overlap, as well as for the same trajectories using a globally-computed database. For each case, we first greedily select the best m locations within each tile, where m is computed according to (4.12). We then greedily select from these tile-computed results to form the global location database. As Figure 5-12a shows, the quality of the location database is highly dependent on the size of the tiles, with large tiles significantly outperforming small tiles. This is because the modified betweenness centrality (3.6) effectively neglects global-scale network structure when computed on subregions. Intuitively, it is generally only beneficial to use limited-access freeways and other high-traffic throughways when traveling long distances, and computing the betweenness centrality for sub-regions confines the computation to short routes. Nevertheless, when sufficiently large tiles are used, such as 1 square kilometer for Moscow, we can still select a valuable location database.

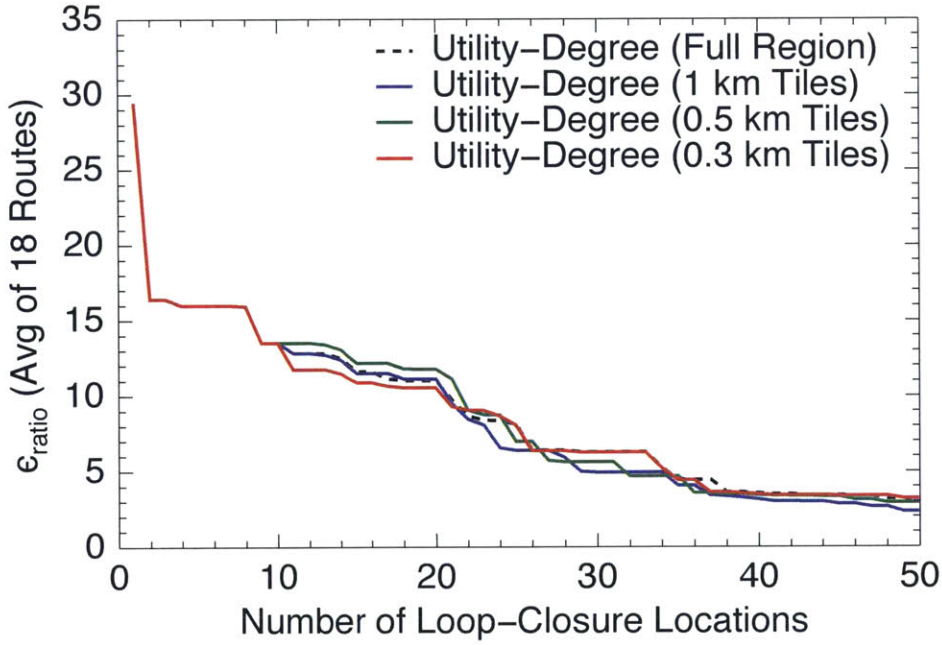
Rather than using betweenness centrality to compute $P(\text{view})$ for tiles, we can instead use the weighted degree centrality, introduced in Section 4.3.1 and demonstrated in Section 5.3.1. The weighted degree centrality provides a suitable, computationally-efficient replacement for betweenness centrality in the greedy location selection algorithm, and is computed entirely locally and independently for each intersection in the street network. As shown in Figure 5-12b, the navigation performance of the selected location databases is independent of the size of the map subregions used to compute them. The difference is subtle, but after about 40 locations have been selected (1.4% of the full database), the databases selected using degree centrality outperform even those computed using one square kilometer tiles and betweenness centrality.

The results in Figure 5-12 show that the overall navigation utility of databases computed using hierarchical selection can approach that of those computed globally for the entire map. This enables the extension of maximum-utility database selection to much larger regions than the global-scope greedy optimization. Further, it allows

efficient re-computation of the location database if a portion of the street network changes, such as due to road construction, by only re-computing the $P(\text{view})$ and location selection for the affected tiles. If the best m locations in the tile change after the update, the global selection should then be re-computed.



(a) $P(\text{visit})$ from betweenness centrality



(b) $P(\text{visit})$ from degree centrality

Figure 5-12: Comparison of average pose uncertainties when navigating through Moscow using location databases selected with the efficient tile-based hierarchical approach. The best navigation performance is achieved when using large tiles and approximating $P(\text{visit})$ using the modified betweenness centrality within the tile's boundaries. However, smaller tile sizes, which result in more efficient computation, benefit from instead using weighted degree centrality to approximate $P(\text{visit})$, which is computed locally for each street intersection and is thus independent of the tile region size.

5.5 Chapter Summary

This chapter applied maximum-utility location selection to construct limited-size loop-closure databases for vehicle navigation in network environments. Section 5.1 introduced a suite of open-source tools developed to route a vehicle through a street network (`OpenStreetMap.jl`) and generate a pose-graph from simulated odometry and loop-closure measurements (`PoseGraphSimulation.jl`).

Section 5.2 demonstrated maximum-utility location selection for several major cities with varying topologies. These qualitative results showed that the maximum-utility locations in a street network tend to be intersections that are frequently visited but well-dispersed. Section 5.3 provided quantitative evaluations of reduced-database performance, demonstrating that accurate trajectory estimates can be computed even when using greatly-reduced location navigation databases. We additionally used the vehicle simulator to aggregate the results of tens of thousands of simulated pose-graphs of long-duration trajectories for reduced-database performance evaluation, showing that maximum-utility location selection out-performed all alternative database selection approaches tested.

Section 5.4 showed that well-performing reduced navigation databases can be selected efficiently for very large regions using a tile-based hierarchical location selection approach. The measurement probability at each street intersection can additionally be approximated using its weighted degree centrality (4.13) to achieve tile-size-independent location selection results, further reducing the computational requirements of maximum-utility navigation database selection.

Chapter 6

Terrain Relative Navigation

Terrain Relative Navigation (TRN) systems provide vehicle position and orientation updates by correlating sensor data, such as from a scanning lidar or camera, with a pre-determined database of globally-referenced terrain during mobile vehicle operation. As Section 6.1 will discuss, TRN systems are considered essential for extraterrestrial pinpoint landing from orbit or GPS-denied unmanned aerial vehicle (UAV) navigation, as they provide high-precision, drift-free absolute position and attitude (pose) measurements. This capability is similar to that provided by a GPS receiver and a star camera but is applicable in planetary environments lacking GPS coverage or adequate visibility for star tracking.

Similarly to the street map navigation system presented in Chapter 5, the performance of a TRN system is dependent on the contents and quality of its onboard terrain database. In many applications, this database must be computed prior to flight without knowledge of the true flight trajectory, including possible atmospheric disturbances or vehicle actuation. Ideally, the terrain database would contain a highly-detailed model of the entire operational environment, however, autonomous flight vehicles typically have very limited onboard storage, computational resources, and communication bandwidth. To overcome these limitations, the terrain database can instead consist of a collection of *landmarks*, which represent distinct, recogniz-

able terrain features. These landmarks might consist of image or elevation patches in high-contrast areas or higher-level features, such as craters. However, the collection of all possible landmarks in the environment is often still too large to store onboard the vehicle, effectively search within during real-time operation, or transmit over low-bandwidth communication networks.

In this chapter, we address the case of vehicles operating in 3-dimensional, pathless environments, thus relaxing the earlier assumption from Chapter 5 that the vehicle is operating in a 2-dimensional, network environment. Landmark-based TRN systems typically use either a camera or lidar as the terrain sensor, which means the landmark observations take the form of line-of-sight measurements. We use the landmark utility metric from Section 3.3, which extends the location utility metric to account for line-of-sight measurements, to determine the N best landmarks in the flight vehicle's operational environment prior to flight and construct the optimal terrain database for vehicle navigation. Landmark utility is defined independently of any specific trajectory, instead depending on the probability that the landmark is observed given a probabilistic representation of the predicted vehicle trajectory and the spatial distribution of the landmark with respect to other landmarks in the database.

We use the limited prior information available from Monte Carlo trajectory simulations and a low-fidelity model of the environment (such as from satellite imagery) to compute our landmark measurement probability and spatial distribution vectors, and can then use the techniques described in Chapter 4 to (approximately) maximize the utility of the flight computer's onboard terrain database. This utility-based landmark database selection approach significantly improves TRN performance over either randomly selecting landmarks for database inclusion or evenly spacing them on a grid for a fixed-size database, and is more efficient than manually selecting landmarks. This selection approach is applicable to any landmark-based TRN system that uses line-of-sight measurements (including camera- and lidar-based systems).

In order to demonstrate our database selection algorithm, we additionally introduce a new, generally applicable TRN framework utilizing a factor graph representation and an incremental smoother in Section 6.2. This TRN system does not suffer from the linearization errors of existing filter-based approaches and allows for incorporation of both landmark observations and opportunistic terrain-feature tracking, as well as other potentially available sensor data, such as that of inertial measurement units (IMUs), GPS receivers, altimeters, and star cameras. We demonstrate our TRN algorithm and utility-based landmark selection in the context of the Terrain Relative Navigation & Descent Imager (TRNDI) system [207], an optical payload designed for Draper Laboratory's GENIE (Guidance Embedded Navigator Integration Environment) guidance, navigation, and control (GNC) system. GENIE is capable of emulating planetary approach and landing trajectories using a terrestrial test rocket [208, 209, 210]. Section 6.3 provides details of the GENIE GNC system, the TRNDI payload, and the TRNDI imagery and TRN simulators.

In the remainder of this chapter, we demonstrate landmark-based TRN using limited-size landmark databases computed using the greedy algorithm proposed in Section 4.2. Section 6.4 describes the computation of the landmark viewing probability, $P(\text{view})$ using GENIE's Monte Carlo flight simulation data generated during safety-of-flight tests. We have been unable to fly the TRNDI payload onboard a GENIE test flight. Instead, we here use the most recent GENIE test flight trajectory and the TRNDI simulators to evaluate the performance of both the utility-based landmark selection algorithm (Section 6.5) and the graph-based TRN solver (Section 6.6). Much of the results presented in this chapter were previously published in [211].

6.1 Related Work

The majority of planetary landing systems to date, including the recent Mars Science Laboratory built by NASA's Jet Propulsion Laboratory (JPL) [212], have used a Doppler radar to sense vehicle range to the surface and surface-relative velocity. NASA's Autonomous Landing Hazard Avoidance Technology (ALHAT) Project, which is currently developing terrain relative navigation and hazard avoidance capabilities for use with future high-precision manned landing missions, favors a combination of a Doppler and flash lidars [213, 208]. However, both radar and lidar options have high mass and power requirements for unmanned missions, and thus much recent work has focused on development of vision-based systems to serve as potential replacements [214, 215, 216, 217, 218]. Cuseo, et. al first described a machine vision-based approach for "recognition of landmarks to reduce navigation errors and achieve a more precise landing" [219]. Shortly after, Liebe described a method to use terrain feature tracking for vehicle attitude estimation [220].

More recently, the JPL VISINAV system used a variant of an Extended Kalman Filter (EKF) to process both visual and inertial measurements for landing navigation [221, 222, 223]. This filter, called the Multi-State Constraint Kalman Filter (MSCKF), maintained multiple vehicle poses in its state vector in order to accommodate tracking opportunistic features in addition to pre-mapped landmark measurements in a consistent manner [224, 30]. This system was successfully tested on a parachute dropped from a sounding rocket, achieving a landing position estimation error of less than 7 meters [225]. The European Space Agency (ESA) and its partners have developed a similar system, which was demonstrated using a robotic arm and a large, simulated lunar surface [226].

Singh and Lim showed that TRN accuracy could be improved by adding the positions of tracked landmarks into the EKF state vector [227]. The landmark database was used to provide the initialization for the landmark location rather than an absolute truth position, and the landmarks were incorporated into the state vector upon first observation, allowing their positions to be successively refined and effectively achieving sub-pixel tracking resolution within the prior map.

Similar to the MSCKF approach, Sibley, et. al [81] use a Sliding Window Filter (SWF) to maintain a constant-length window of poses in the estimator state. As opposed to the MSCKF, the SWF performs bundle-adjustment within the window to estimate an elevation map of the landing site for hazard detection during the final stages of vehicle landing. However, the SWF does not incorporate pre-mapped terrain information, and thus can only localize the vehicle within a local coordinate system.

Johnson and Montgomery provide a survey of ten different vision and lidar-based TRN approaches [228]. They subdivide these approaches into active ranging vs. passive imaging and by absolute (global) vs. local position estimation. Absolute position enables active guidance toward a pre-specified landing site, while local position is capable of higher precision and is therefore more useful for hazard avoidance. They conclude that because local precision does not involve a direct comparison to a global reference map, these measurements cannot be used to improve absolute precision. They further state that multiple local maps must be generated for different altitudes, due to differing resolutions, which contributes to local position estimation error.

In the following section, we present a new TRN approach that utilizes recent developments in Bayesian factor graph-based optimal smoothing [229, 67, 68], rather than the fixed-length or windowed filtering approaches previously used for TRN. Our approach extends upon that of [227] by incorporating landmarks into the smoother using a location prior as both an initialization point and a measurement, which anchors the estimator in the global coordinate frame while also allowing opportunistically tracked features to be added into the landmark database and estimated online. Unlike the

MSCKF, the smoothing approach allows opportunistic feature measurements to be immediately incorporated and utilized by the filter after only two observations, rather than waiting for them to exit the sensor field of view. Furthermore, we eliminate the need to construct multiple local maps by maintaining all measurements and poses during the approach and landing trajectory. We instead keep all poses in a single, consistent reference frame, overcoming the limitations predicted by [228]. This TRN approach is additionally capable of fusing measurements from other commonly available sensors on spacecraft or UAVs, such as IMUs, altimeters, star cameras, or GPS receivers, in a common framework.

6.2 Graph-based TRN Solver

In this section, we introduce a factor-graph-based incremental smoothing approach for TRN, which provides a common framework for optimization-based sensor fusion problems. This approach builds upon recent work in the field of Simultaneous Localization and Mapping (SLAM) for mobile robotics. Factor-graphs provide a unified perspective to summary-propagation algorithms, in which "messages" are passed along the edges of the graph, which include probability-propagation algorithms such as the Kalman filter and optimal smoothing [230].

A factor-graph is a bipartite graph consisting of both factor and variable nodes. Variable nodes represent unknowns in the problem, such as the vehicle's pose. Factor nodes represent probabilistic constraints on the variables, each of which is modeled as an error function to be minimized. (6.1) defines a generic factor $\mathcal{F}_i$ that relates variables v_k^i and measurement z_i through an error function, $err(v_k^i, z_i)$, and a cost function, $d(\cdot)$. Superscript i here refers to all variable nodes constrained by the factor $\mathcal{F}_i$, and k is the current time-step.

$$\mathcal{F}_i(v_k^i) = d(\text{err}_i(v_k^i, z_i)) \tag{6.1}$$

For example, (6.2) shows a factor for a measurement with additive Gaussian noise and a nonlinear measurement model, where z_i is the measurement, $h(v_k^i)$ is the nonlinear measurement function, and Σ is the noise covariance matrix.

$$\mathcal{F}_i(v_k^i) = \exp\Big\{ -\frac{1}{2}||h(v_k^i) - z_i||_\Sigma^2 \Big\} \tag{6.2}$$

Maximum a-posteriori (MAP) inference is straightforward when using a factor-graph representation, as it only involves maximizing the product of the factor nodes [231]. This has led to the popularization of the factor-graph representation for solving localization and SLAM problems. In the remainder of this section, we will first model the TRN problem using a factor-graph representation (similar to the pose-graphs used in Chapter 5) and then perform inference on this graph to estimate the vehicle trajectory using the iSAM2 solver [67, 68].

6.2.1 Graphical Representation of TRN

Terrain relative navigation is well-modeled as a factor-graph, an example of which is shown in Figure 6-1. The primary optimization variables of interest are the vehicle poses, $\boldsymbol{x} = \{x_0, \ldots, x_N\}$. We additionally estimate the locations of the pre-mapped landmarks, $\boldsymbol{\ell} = \{\ell_0, \ldots, \ell_M\}$, to improve upon their associated location priors, $\boldsymbol{p_\ell} = \{p_{\ell_0}, \ldots, p_{\ell_M}\}$, whose precision corresponds to the resolution of the terrain database. We can optionally estimate the locations of opportunistically tracked (unmapped) terrain features, $\boldsymbol{f} = \{f_0, \ldots, f_J\}$, to effectively add new landmarks to the database. Adding measurements to the graph simply involves incorporating additional edge constraints (factors) between these pose, landmark, and feature variables. Every factor is phrased as an equation of some subset of these variables and is weighted by its respective measurement noise covariance, as in (6.2).

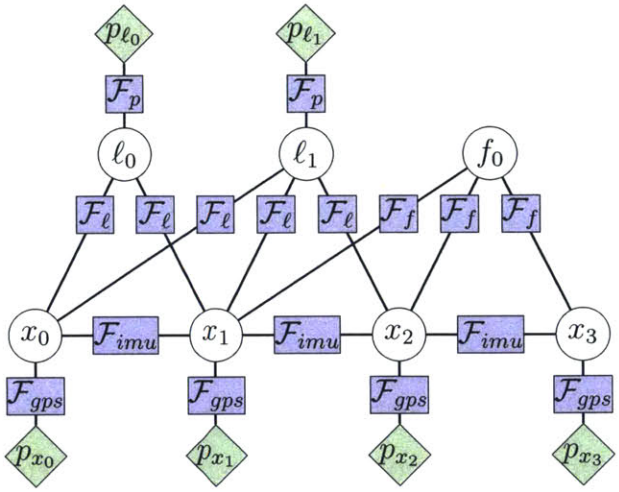


Figure 6-1: Example graphical model for terrain relative navigation. Variables are shown as white nodes, factors (constraints) as blue boxes, and constant priors and absolute-frame measurements as green diamonds. Consecutive vehicle poses x are connected by IMU factors, and landmarks ℓ and opportunistic features f are connected to poses by projection factors $\mathcal{F}_\ell$ and $\mathcal{F}_f$. If available, global-frame measurements $\mathcal{F}_{gps}$, such as from a GPS receiver or star camera, can also be connected to poses. The only difference between the handling of opportunistic features and landmarks is the additional location prior provided by the landmark database, p_{ℓ_i}.

Most TRN measurements can be modeled using one of the three basic types of measurement factors defined by [63]. The first class are unary measurement factors, which handle external measurements to an absolute frame, including prior location information. We can use unary factors to model global-frame measurements, such as from a GPS receiver or star camera, if available. Unary factors are also used to model the location prior on mapped landmark locations, which have an associated uncertainty due to the limited resolution of available satellite imagery and other maps. The second class of factors encode relative-pose measurements that fully constrain the relationship between two poses, such as from an IMU, visual odometry system, or a probabilistic vehicle motion model (or combination thereof). The third class of factors represent measurements that partially constrain (i.e., are low-rank) the relationship between either two poses or a pose and a terrain feature, such as line-of-sight observations of landmarks or opportunistic features. Projective camera measurements are low-rank, and thus camera-based feature observations can constrain only two of an opportunistic feature's three position variables. This means that at least two ob-

servations of an opportunistic feature are required before the feature can be added to the graph. Landmark observations can be incorporated into the graph immediately because every landmark, ℓ_m, has an associated location prior, p_{ℓ_m}, from the terrain database that provides a full-rank position constraint. These three classes of factors can model any pose-pose, pose-landmark, and pose-feature constraints, as well as global-frame pose or landmark measurements.

6.2.2 Factor Graph Optimization

Factor-graph models for TRN are inherently sparse because factors relate only consecutive poses or pairs of poses and landmarks or features, resulting in a highly-structured factor-graph topology. This means we can solve the factor-graph optimization problem efficiently using iSAM2 (incremental Smoothing and Mapping), an incremental solver that exploits this type of sparse graph structure [67, 68]. We additionally use the RISE algorithm [65], an incrementalized version of the Powell's Dogleg [232, 233] numerical optimization method, to enhance the robustness of the solver to objective function nonlinearity and numeric ill-conditioning. These solvers have been conveniently packaged in the open-source GTSAM library [231], along with numerous factor constraint model definitions.

The iSAM2 solver allows the solution to be efficiently updated incrementally. This means that only the past poses that are affected by the most recent measurements are updated at any given time, making it computationally efficient to maintain the entire trajectory rather than performing marginalizations, as in the Sliding Window Filter [81] or Multi-State Constraint Kalman Filter (MSCKF) [221]. Because previous poses are also updated by the smoother, past linearization errors can be corrected to improve the real-time estimate of the current pose as compared to that of an EKF. Recent work on robust SLAM algorithms has also led to approaches for recovering from incorrect data association in factor-graph-based solvers [234, 235, 236, 237]. In the case of TRN, where the trajectory typically will not "loop" back upon itself, the computational cost of updating these past poses when necessary is typically small,

and the incremental solver could be implemented on a constrained system, such as TRNDI. Because only the measurement constraints need to be maintained by the graph rather than the actual sensor data, maintaining these past poses does not invoke a significant storage penalty in comparison to the landmark database, which stores detailed terrain descriptors to enable data association. Furthermore, as the flight trajectory is observed, landmarks determined to have low or zero utility could be removed from the database to make room for new factors and opportunistic features. By combining the factor-graph representation with the iSAM2 factor-graph solver, we can efficiently compute trajectory estimates incrementally in real-time for terrain relative navigation.

6.3 Flight Simulation Framework

The Guidance Embedded Navigator Integration Environment (GENIE) is a guidance, navigation, and control (GNC) system developed by Draper Laboratory to mature and demonstrate technologies that can perform safe and precise planetary landings. Since March 2012, 20 GENIE closed-loop flights [209, 210] have occurred onboard the Masten Aerospace Xombie terrestrial test rocket [238]. Figure 6-2 shows GENIE and Xombie during one of these closed-loop terrestrial flight tests. Figure 6-3 shows the trajectory of the GENIE Campaign 5 test flight, which was conducted in March of 2013 and consisted of a 500 meter ascent followed by a sloping descent emulating a planetary landing trajectory, with a total downrange traversal of 300 meters.

Section 6.3.1 provides an overview of GENIE and its navigation sensors and discusses their current limitations. To work towards overcoming these limitations, we developed the TRNDI optical payload, described in Section 6.3.2, to serve as a vision-based navigation and guidance algorithm testbed. Section 6.3.3 details the TRNDI TRN simulator, which was developed for TRN algorithm development and pre-flight testing in preparation for a future flight test.

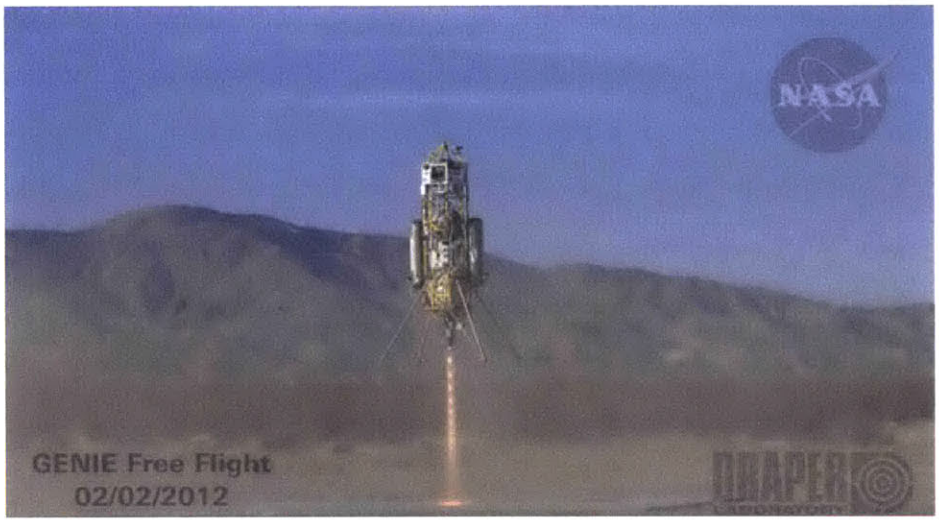


Figure 6-2: Draper Laboratory's GENIE GNC system onboard the Masten Space Systems Xombie terrestrial test rocket performing a closed-loop flight test.

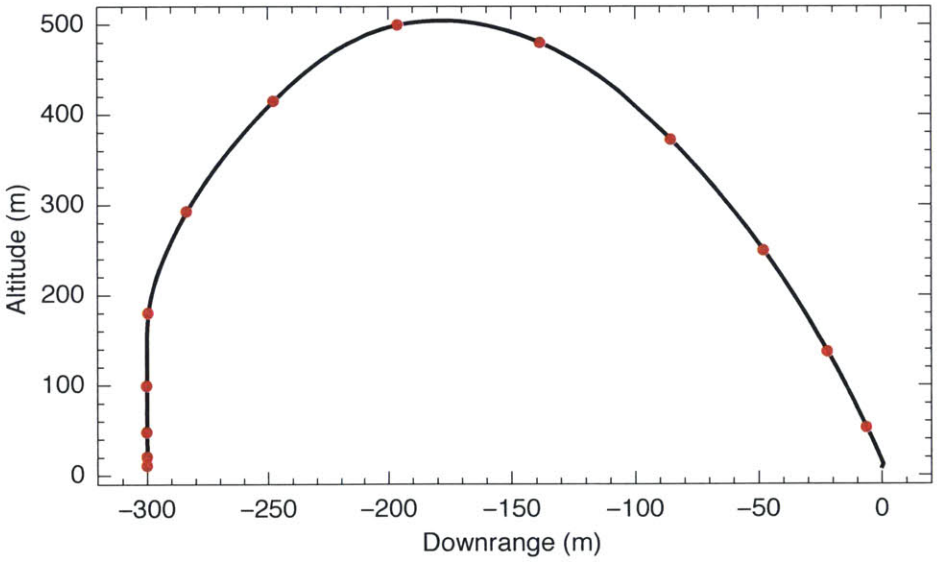


Figure 6-3: The GENIE Campaign 5 flight trajectory, as estimated by GENIE's onboard navigation system. The red dots are spaced every 5 seconds of flight time.

6.3.1 GENIE GNC System

The Draper Laboratory-built GENIE (Guidance Embedded Navigator Integration Environment) system is a self-contained hardware and software avionics platform capable of real-time autonomous guidance, navigation, and control (GNC) in a rugged terrestrial environment [209]. The GENIE project aims to extend closed-loop flight capability of a NASA-sponsored terrestrial test rocket to help validate future NASA payloads seeking Lunar- or Mars-like approach trajectory environments here on Earth. Additionally, through successful rocket flights, GENIE seeks to mature and demonstrate Draper GNC technology that can perform safe and precise planetary landings applicable to a broad range of NASA missions.

GENIE's sensing suite consists of a Northrup Grumman LN200 IMU to measure vehicle translational and rotational rates, a Javad GPS to measure vehicle position and velocity, and an Acuity Laser altimeter to measure vehicle altitude. While the success of the various flight-testing campaigns has proven this combination of sensors to be sufficient for terrestrial test flights [209, 210], the system has two weaknesses applicable to non-terrestrial planetary flight. First, dependency on GPS limits GENIE's ability to simulate realistic planetary approach and landing navigation, as GPS is unavailable in the actual operational environment. Second, there is a lack of redundancy in the measurement of vehicle attitude. While the GPS is used to constrain the position and velocity measurements integrated from the IMU's accelerometers, there is no easily-available daytime star camera analog to constrain the attitude measurements integrated from the IMU's gyroscopes. This could become a limitation in either the case of a particularly long flight (short GPS outages are acceptable) or a sensor failure in flight.

These limitations are not unique to GENIE and have been the subject of much research in recent years, as was discussed in Section 6.1. Several sensing systems have been proposed to fill this gap, but the vast majority involve radar, lidar, or optical cameras, with camera-based TRN systems being an especially active and promising

area of research [228, 214, 215, 216, 217, 227]. However, to our knowledge, no such vision-based navigation system has yet been demonstrated on a powered Lunar- or Mars-like approach and landing trajectory. The GENIE system, with its demonstrated ability to perform accurate planetary trajectory simulations onboard a terrestrial test rocket, is particularly well-suited to perform algorithm validation in a relevant, powered-flight environment.

6.3.2 TRNDI Optical Payload

The Terrain-Relative Navigation & Descent Imager (TRNDI) is a self-contained, low mass and power prototype spacecraft payload providing vision-based navigation and guidance for safe planetary approach and landing. TRNDI is scheduled to be included on multiple upcoming GENIE flights in order to demonstrate a suite of vision-based algorithms' capabilities and robustness. This section details the algorithms, software framework, and hardware and sensor systems that make up the TRNDI system.

TRNDI includes three classes of vision-based algorithms relevant to navigation and guidance during planetary approach and landing: Terrain-Relative Navigation (TRN), Visual Odometry (VO), and Hazard Detection (HD). TRNDI also includes the GENIE Initial Direction Enhancer (GIDE), which is an automated tool for terrestrial rocket navigation initialization. TRNDI's HD and GIDE algorithms and their pre-flight simulation results are discussed in detail in [207]. For purposes of this thesis, we will focus on TRNDI's TRN system.

The Terrain Relative Navigation (TRN) system provides globally-referenced vehicle attitude and position updates by identifying unique, pre-mapped landmarks along the planned vehicle flight path. Points of interest are identified in the camera images during flight and matched to a catalogue of known landmarks created from satellite imagery prior to flight. The TRN system provides functionality similar to a GPS receiver and star camera in planetary environments lacking GPS coverage and adequate

visibility for star tracking. However, the performance of TRN systems is proportional to the resolution of their *a priori* terrain maps, as only distinctly resolvable objects and textures can be identified and catalogued as landmarks. These limitations become increasingly apparent at low altitudes.

TRNDI uses a modularized C++ software framework intended for real-time asynchronous data collection, processing, and logging [239]. This framework encapsulates each sensor and algorithm within a "task," passing data between tasks using a common queue in the form of standardized "messages." Algorithms and sensors can be easily added or removed by simply adding or removing their associated tasks from the message queue. These tasks run asynchronously in separate threads, making the system inherently multithreaded and allowing sensors and algorithms to be independently triggered by data availability, maximizing use of the CPU.

TRNDI's hardware comprises an embedded computer, a dedicated battery, and three dedicated cameras, which are shown sitting on a lab bench in Figure 6-4. The computer is a rugged, EMI-shielded IDAN RTD computer with an Intel Core 2 Duo processor, 4 GB of RAM, and a 480 GB internal SSD for long-duration data logging, and runs Ubuntu Server 12.04. A lithium polymer battery provides power to the IDAN's internal power regulator for several hours of continuous operation. A ruggedized StarTech USB hub mounted to the computer provides sufficient connections and power for the three cameras.

TRNDI uses compact Edmund Optics USB CCD cameras (EO-1312M) with 1280 $\times$ 1024 resolution, global shutters, and harsh environment packaging. Two of these cameras point 30 degrees (re-configurable) outward from the rocket[1] – one in the forward direction and the other to the left side of the vehicle – providing a balance between maintaining as much terrain in the field of view as possible, observing the terrain nearest to the rocket (typically being directly downward), and avoiding occlu-

[1]Previous results of TRNDI's hazard detection system [207] had the cameras configured to point 45 degrees outward from the rocket, as they were conducted in preparation for GENIE's Campaign 6 test flight. The camera angle is selected according to the planned trajectory profile.

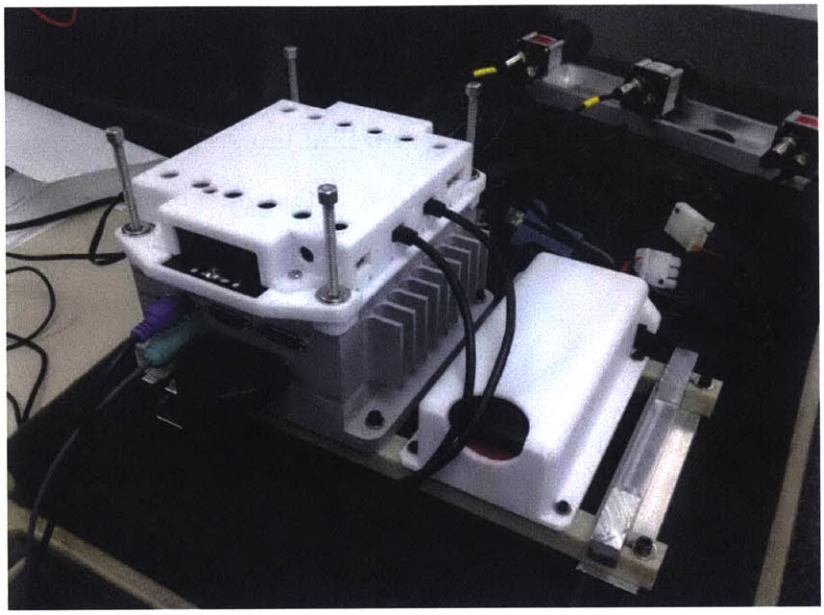


Figure 6-4: The TRNDI flight hardware includes a single-board computer, three USB cameras, and a LiPo battery, shown here sitting on a lab bench.

sions from the vehicle and rocket plume during takeoff and landing. The cameras are hardware-synchronized to 7.5 frames per second (fps) and use Edmund Optics 6 mm compact fixed focal length ruggedized lenses (approx. 60 degree field of view), which are calibrated prior to flight.

TRNDI is capable of participating in GENIE's existing hardware-in-the-loop (HITL) Monte Carlo simulator, which is used to validate the GENIE system prior to each test flight under varying flight conditions. TRNDI interfaces with GENIE over Ethernet during these simulated runs, just as it does during an actual flight. GENIE sends flight data to TRNDI at 50 Hz, including the current best navigation estimate. After each simulated flight, the GENIE simulator sends a command to fully power cycle TRNDI, taking the hardware through its full operational cycle.

TRNDI requires only one additional software module for simulated flights, which generates simulated camera images based on GENIE's most recent navigation state.[2] The image simulation module projects a satellite image of the flight range into the camera view using a homography transform. The image of the flight range is a mosaic of many images from Google Earth stitched together using the Microsoft Image Compositing Editor.

6.3.3 TRN Simulator

The TRNDI TRN simulator generates the odometry measurements, landmark priors, and landmark line-of-sight observations required by the graph-based TRN solver described in Section 6.2. Odometry constraints represent a vehicle motion model or IMU measurements to constrain the relationship between consecutive vehicle poses and ensure that the poses in the factor-graph are fully connected. This connectivity is necessary to keep the solver numerically stable in cases when insufficient landmark observations are available to fully constrain a pose. Landmark priors, which provide a prior distribution on the positions of landmarks in the terrain database in the global frame, are assumed to have a position standard deviation of 0.4 meters, which is approximately the available resolution of Mars imagery. We additionally add a prior on the first pose's position and attitude, which represents the hand-off from an orbital navigation system to the TRN system or, in GENIE's case, prior knowledge of the vehicle's launch site.

An example simulated camera image with line-of-sight observations of terrain landmarks is shown in Figure 6-5. Candidate landmarks for potential inclusion in the terrain database are generated by placing points either randomly or evenly-spaced on visible surfaces throughout the flight-test region. In practice, these landmark candidates might be selected using a visual feature detector to identify easily trackable terrain features, such as the Harris corner detector [83]. For purposes of simula-

[2] A future system upgrade will allow TRNDI to receive truth data directly from the simulator for image generation rather than using GENIE's state estimate.

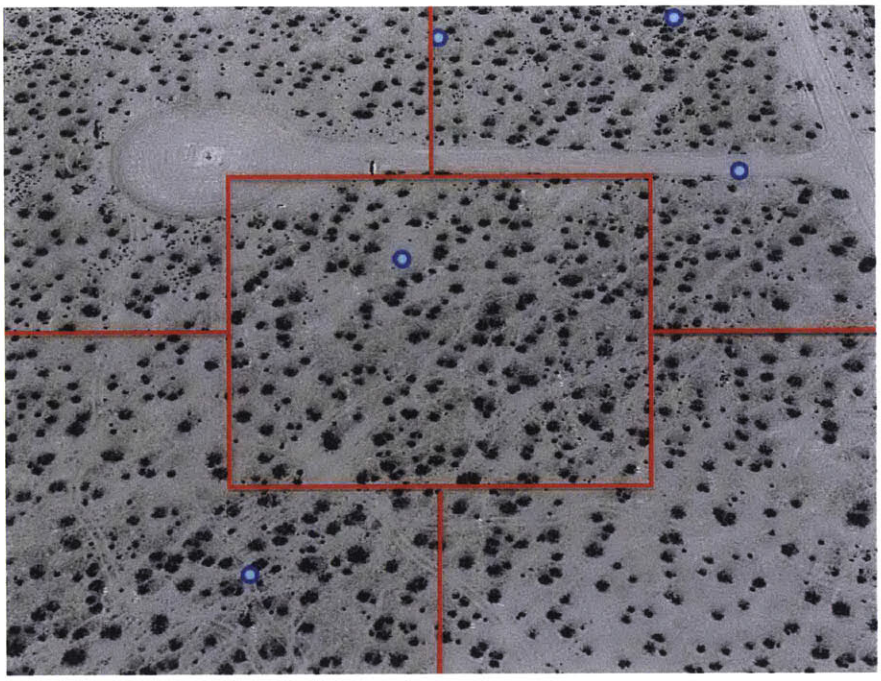


Figure 6-5: Simulated image showing sample landmark observations (blue) and the five regions (red) used to compute observation dispersion in (6.5). This image has three regions containing at least one observation.

tion, we assume accurate feature detection and data association for all tests, as these issues are environment-dependent and have already been well-addressed in the literature. This assumption allows us to generate the set of candidate landmarks on a regular 20×20 meter grid and set the landmark recognition probability in (3.9) to be uniform across all landmarks. Setting a uniform recognition probability and selecting landmarks from a regular grid allows us to observe the most desirable landmark database geometries that result from the various selection approaches without any environment-specific effects or loss of generality. Existing feature tracking methods, such as those presented in [221, 226, 227], could integrate with our presented algorithms without modification.

For the purposes of the experiments presented in Section 6.6, the simulations were run on a 2011 MacBook Pro laptop running Ubuntu 14.04 rather than the TRNDI flight system. The TRNDI TRN simulator generated a text file of measurement constraints, which were then parsed using a custom interface to the GTSAM library, rather than

fully implementing the TRN solver within the flight code. *All* measurements common to the trajectory estimates computed using multiple landmark databases (including all odometry measurements and landmark priors) were generated simultaneously with the same random noise values to allow direct comparison of their results.

6.4 Estimating Landmark Observation Probability

Maximum-utility landmark database selection (4.3) requires the probability of the vehicle's sensors viewing various terrain features (3.9). This probability distribution, $P(\text{view})$, is computed by integrating over the full trajectory space (3.10) in order to approximate the probability of each location in the map being viewed using the trajectory marginalization process described in Section 3.3. However, because there are infinitely many possible trajectories, we instead estimate $P(\text{view})$ by sampling from the trajectory space.

In preparation for the GENIE Campaign 5 flight test in March, 2013, 203 flights were simulated under Monte Carlo flight conditions to evaluate the GNC system performance across the full flight envelope for safety-of-flight [210]. We used these simulated flight trajectories to sample the trajectory space in order to approximate the probability of each location in the terrain map being viewed. We first generated a grid across GENIE's entire operational environment. Then, for every 1 second of flight for each of the 203 trajectories, we counted how many times each grid cell was at least partially viewed by the camera. The estimated viewing probabilities computed using a 5 meter grid spacing are shown in Figure 6-6. Because the pre-flight Monte Carlo simulations are intended to ensure the full flight regime is safe rather than explicitly modeling the expected flight conditions, this collection of flight trajectories only approximates the true viewing probability distribution of the environment.

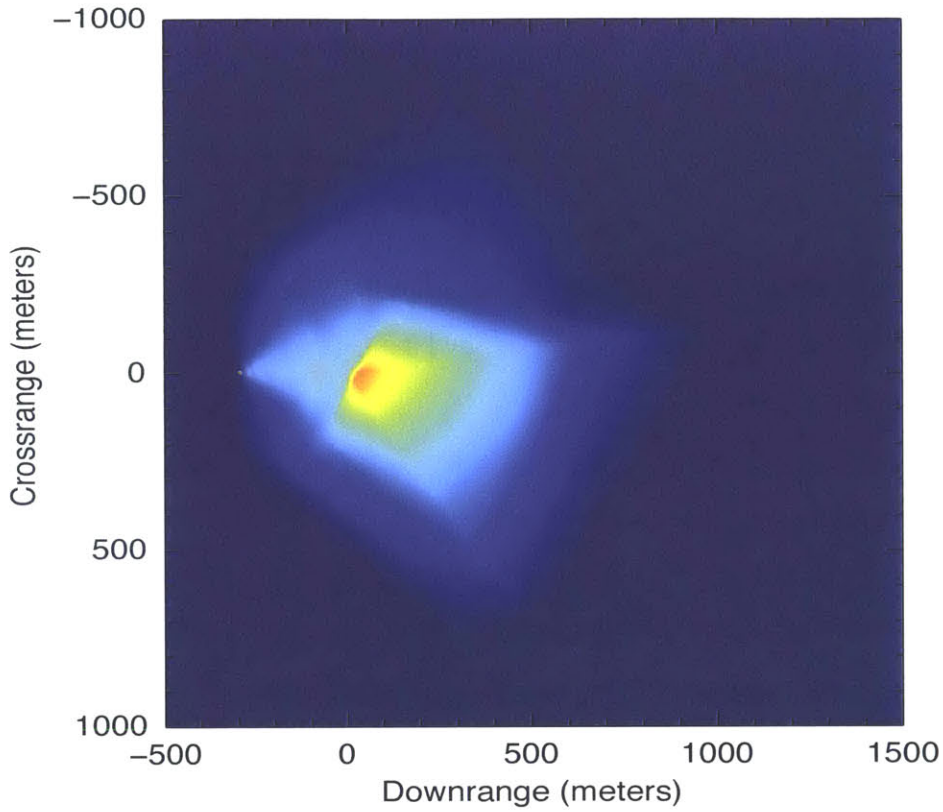


Figure 6-6: Terrain view coverage distribution empirically estimated using 203 Monte Carlo flight simulations generated as part of GENIE's pre-flight testing, normalized by the maximum viewing probability for display purposes. The planned flight trajectory begins at (-300,0) and the landing target is (0,0).

The GENIE Campaign 5 flight trajectory is highly constrained at both the take-off and landing sites because the vehicle always launches from the same position and attempts to land precisely at a set target. Therefore, the camera views these terrain regions during every flight simulation and typically does so for comparatively long durations due to the low vehicle velocity during launch and landing. This results in very high terrain viewing probabilities near both the trajectory start and finish locations in comparison to the typically much lower values elsewhere in the terrain. In order to account for these regions of high viewing probability during maximum-utility landmark selection in the next section, we will slightly increase the utility weighting parameter to $\lambda = 1.2$, which slightly favors well-dispersed landmarks over frequently viewed landmarks. $\lambda = 1.2$ was selected because it empirically gave a

good fit, and the specific value of λ was found to be insensitive in the range of 1.1 to 1.3. This helps to select landmarks that are valuable during the high-altitude portions of the trajectory when the vehicle is moving quickly and its motion is less constrained.

6.5 Results of Landmark Database Selection

This section evaluates the greedy landmark selection approach of Section 4.2 using database evaluation metrics independent of any particular solver, emphasizing the general applicability of utility-based landmark selection to *any* landmark-based TRN solver. Section 6.5.1 first introduces four metrics of landmark database value for navigation. These metrics can be computed without solving for a vehicle trajectory estimate, making them applicable to any landmark-based TRN solver, including the Extended Kalman Filter-based approaches common in the literature. Section 6.5.2 then compares maximum-utility landmark selection against three alternative landmark selection approaches using these evaluation metrics. Section 6.6 will evaluate these same landmark databases using trajectory estimation results computed using the graph-based TRN solver described in Section 6.2.

6.5.1 Landmark Database Evaluation Metrics

We now define four generalized landmark database evaluation metrics that are independent of any particular TRN solver. These metrics are inherently trajectory-dependent, and thus are unavailable at the time of landmark database selection. However, given the trajectory, placing landmarks to directly maximize these objectives in some combination would result in the "true" optimal landmark database, and therefore we can use them to compare the results of pre-flight landmark selection approaches.

First, we want to maximize the total number of measurements, which we quantify using the average number of landmark observations per image, z_{avg}:

$$z_{avg} = \frac{1}{K} \sum_{k} \sum_{i} z_k^i \tag{6.3}$$

Where z_k^i is the i^{th} landmark measurement in image frame k, and K is the total number of frames. Assuming correct data association and error modeling, landmark line-of-sight measurements always have non-negative information content, so incorporating additional measurements typically results in better navigation precision. Handling extra measurements is more computationally expensive in the solver, but it is straightforward to perform measurement down-selection during flight, if necessary.

The second evaluation metric is the fraction of images that contain landmarks in their field of view, K_{obs}:

$$K_{obs} = \frac{1}{K} \sum_{k} m(\boldsymbol{z}_k) \tag{6.4a}$$

$$m(\boldsymbol{z}_k) = \begin{cases} 1 & |\boldsymbol{z}_k| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.4b}$$

Function $m(\boldsymbol{z}_k) = 1$ if at least one landmark observation, z, occurred in frame k. We ideally want the landmark measurements to be evenly spread across all images without any gaps, which are effectively wasted processing time during which the error covariance grows without correction.

It is additionally important for landmark measurements to be well-dispersed in the field of view, as this results in a higher information content per measurement, as discussed in Section 3.3. To measure this effect, we segment the camera field of view into five regions of equivalent area, as shown in Figure 6-5. For each image, we count the number of these regions that contain a landmark measurement. The third evaluation metric is then the average number of these populated regions per image

over the full trajectory, r_{avg}:

$$r_{avg} = \frac{1}{K} \sum_{k} R(\boldsymbol{z}_k) \tag{6.5}$$

Function R counts the number of regions shown in Figure 6-5 containing at least one measurement in $\boldsymbol{z}_k$. This evaluation metric measures the dispersion of measurements in the camera field of view without penalizing large measurement counts for a particular frame.

The fourth metric is simply the fraction of landmarks that are actually observed during the trajectory, ℓ_{obs}:

$$\ell_{obs} = \frac{|\boldsymbol{\ell} \in \boldsymbol{z}|}{|\boldsymbol{\ell}|} \tag{6.6}$$

Where $\boldsymbol{\ell}$ represents all landmarks in the terrain database, $\boldsymbol{z}$ is all landmark observation measurements for the trajectory, and $\boldsymbol{\ell} \in \boldsymbol{z}$ is the set of landmarks observed by $\boldsymbol{z}$. Unobserved landmarks are effectively wasted space in the database (though this is only known in hindsight).

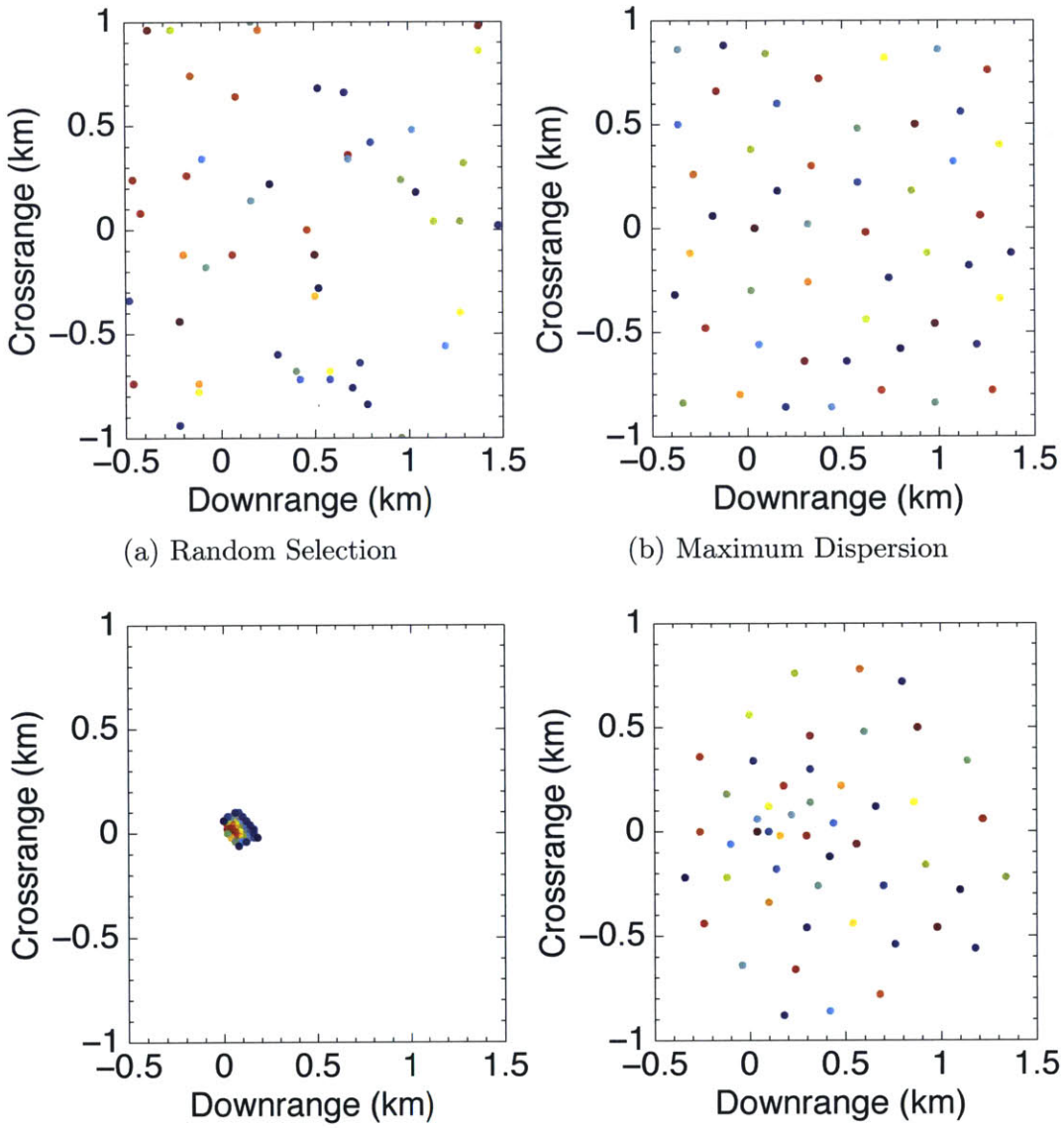
6.5.2 Comparison of Landmark Database Selection Approaches

We now compare the database generated using maximum-utility landmark selection against those generated by three alternative methods using the four evaluation metrics described in Section 6.5.1. Each terrain database contains 50 landmarks, greedily selected from a uniform grid of 10,201 candidate landmarks spaced 20 meters apart in the 2×2 km flight region.

The four landmark database selection methods produce the sets of landmarks shown in Figure 6-7. The first selection method is simply random selection (Figure 6-7a) from a uniform distribution across all candidate landmarks. This roughly emulates selecting the 50 most distinct visual features in the environment, which tend to be clustered in some areas with distinctive terrain features and sparse in others. The second method (Figure 6-7b) attempts to maximize the spatial distribution of the landmarks in the

flight environment, which is equivalent to setting $\lambda \gg 1$ in (4.3). The third method (Figure 6-7c) attempts to maximize the total number of landmark observations over the trajectory by selecting the landmarks with the highest probability of being viewed, equivalent to setting $\lambda = 0$ in (4.3). These landmarks are heavily concentrated near the launch and landing sites, where the trajectory is most tightly constrained. The fourth method (Figure 6-7d) maximizes the total utility of the landmark database. We found $\lambda = 1.2$ to give a good balance between selecting well-dispersed and frequently-viewed landmarks. As discussed in Section 6.4, the weighting parameter was increased from the nominal $\lambda = 1.0$ to compensate for the especially high probability of viewing terrain near the launch and landing sites.

Table 6.1 provides the evaluation metrics computed for each of these landmark selection approaches, computed according to GENIE's actual Campaign 5 flight trajectory (shown in Figure 6-3). The results from random selection represent the average of ten randomly selected databases. Maximizing the landmark spatial distribution increases the average number of image regions containing landmark observations, r_{avg}, but otherwise gives comparable performance to random selection. Maximizing the number of landmark views increases the total number of observations, z_{avg}, database utilization, ℓ_{obs}, and the number of images containing observations, K_{obs}, but reduces the average spatial distribution of observations within each image, r_{avg}. This is because all of the selected landmarks lie within the camera frame during the highly-constrained launch and landing portions of the trajectory, and thus they are very tightly clustered. Despite achieving many observations and a high database utilization, these observations all have very low marginal information content. This means that many of these low-value observations would likely be pruned out in an actual operational scenario to reduce the computational load of the solver using landmark measurement downselection methods such as [240, 241], resulting in a large reduction in z_{avg} and ℓ_{obs} in practice.



(a) Random Selection

(b) Maximum Dispersion

(c) Maximum Observations

(d) Maximum Utility

Figure 6-7: Landmark databases computed by four different *a priori* landmark selection approaches. Landmarks are colored from dark red to dark blue according to the order in which they were selected by the greedy algorithm. The planned flight trajectory begins at $(-300, 0)$ and the landing target is $(0, 0)$.

Table 6.1: Comparison of Landmark Selection Approaches

	Random (Average)	Max. Dispersion	Max. Meas.	Max. Utility
Observations per Image, z_{avg}	1.4	2.0	22.7	5.3
Images with observations, K_{obs}	46%	66%	61%	77%
Avg. populated image regions, r_{avg}	1.2	1.7	1.6	2.4
Database utilization, ℓ_{obs}	18%	24%	100%	52%

Results for 50-landmark databases shown in Figure 6-7 evaluated using the GENIE Campaign 5 test flight trajectory.

The maximum-utility landmark database combines the maximum-dispersion and maximum-observation approaches, yielding improvements over both random selection and maximum-dispersion across all four evaluation metrics. As one would expect, the maximum-observation selection approach results in the largest z_{avg} and ℓ_{obs} values, but these values are highly inflated for a typical system, which would likely throw out most of the observations due to their redundancy. By balancing more potential observations with better spatial distribution throughout the environment, the maximum-utility selection approach most notably achieves a significant improvement in the average number of populated image regions, r_{avg}, as compared all other selection approaches, which will improve vehicle position estimation when using *any* TRN estimator utilizing line-of-sight measurements.

The maximum-utility method outperforms random selection and maximum-dispersion selection on all four metrics and outperforms maximum-observation selection on two of the four metrics. However, as discussed above, the z_{avg} and ℓ_{obs} metrics suffer from diminished returns and are therefore highly inflated for a system performing redundant measurement removal. Therefore, the maximum-utility selection approach is the best overall choice for landmark database selection.

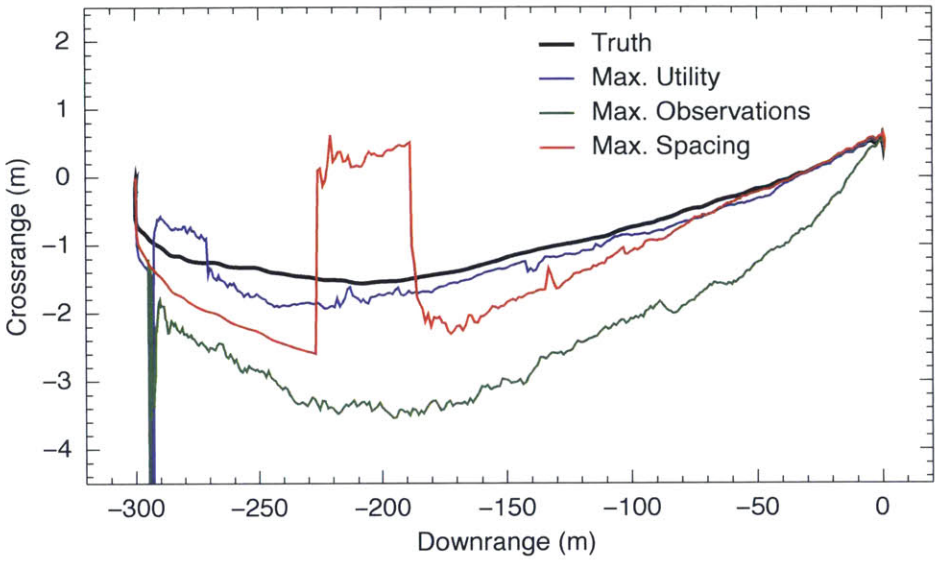
6.6 Graph-based Solver Results

We now evaluate our utility-based landmark selection method using trajectory estimates computed using the graph-based TRN system introduced in Section 6.2. We use the TRNDI TRN simulator described in Section 6.3.3 and the *flown* GENIE Campaign 5 flight trajectory, shown in Figure 6-3, to simulate landmark observations with a 10 Hz camera frame rate. Because the true trajectory is unknown, we must instead simulate using GENIE's own trajectory estimation, and thus we are unable to make a direct comparison to GENIE's current EKF-based navigation system. The process of generating these simulated measurements was discussed in Section 6.3.3. Treating GENIE's trajectory estimate as truth additionally means we cannot use GENIE's actual IMU measurements without inducing a bias, so we instead simulate 6-degree-of-freedom IMU measurements with Gaussian noise to serve as basic odometry measurements in order to guarantee connectedness of the graph. In order to show the applicability to long-duration trajectories and lower-cost sensors, these simulated measurements are noisier than those actually available from GENIE's own onboard IMU and result in a several meter drift when directly integrated without additional sensor measurements.

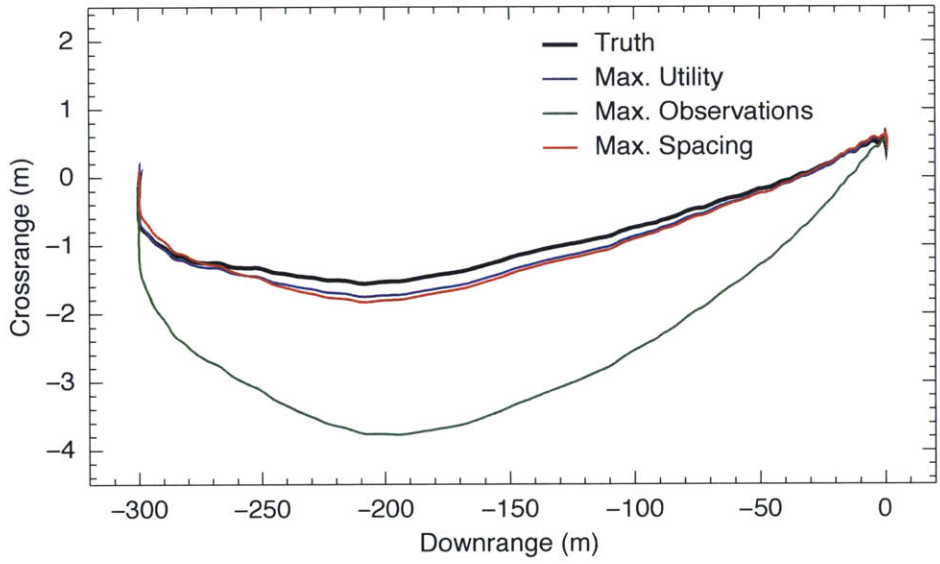
Section 6.6.1 uses the graph-based TRN solver to compare the position error of trajectory estimates computed using the maximum-dispersion, maximum-observations, and maximum-utility landmark databases from Section 6.5. Section 6.6.2 then uses the solver to compare trajectory estimates computed with variously sized maximum-utility landmark databases. The results show that maximum-utility landmark databases outperform alternatively selected databases and achieve high-accuracy trajectory estimates even when using very few landmarks.

6.6.1 Comparison of Selection Methods

We first compare the performance of our graph-based TRN system when using the maximum landmark dispersion, maximum expected landmark observations, and maximum landmark utility selection methods described in the previous section. Figure 6-8a shows a top-down view of the vehicle position estimated in real-time by the solver (i.e., the incremental smoothing result). This real-time position estimate is approximately equivalent to the maximum likelihood smoothed result given all measurements up to that time and represents the estimate of the most recent pose at each time step, which is available to the vehicle for decision-making. Figure 6-8b shows a top-down view of the vehicle position estimated in post processing by smoothing across all poses. Because the graph-based solver is an incremental smoother (as opposed to a filter), the real-time result at the final time-step is equivalent to the smoothed result. Because our graph-based incremental smoother re-linearizes past measurements given more recent information, unlike traditional filtering approaches, these two trajectory estimates are dramatically different. This re-linearization also significantly improves the robustness of the estimator by allowing the incremental smoother to potentially recover from past linearization errors that might otherwise cause a traditional filter to become inconsistent. The maximum-utility database achieves better real-time and smoothed trajectory estimates than the maximum-dispersion or maximum-observation databases.



(a) Real-Time Estimate



(b) Smoothed Estimate

Figure 6-8: Comparison of the (a) real-time (most recent pose incrementally computed) and (b) smoothed (all poses at final time step) position estimates for the GENIE Campaign 5 flight trajectory using three landmark database selection methods. The smoothed result is identical to the final real-time result. The maximum utility selection method results in both the most accurate real-time and smoothed trajectory estimates, and the final position error is about 0.2 meters, or about 0.07% of the downrange flight distance.

Figure 6-9 shows the real-time position error of the terrain estimator for the duration of the flight when using the three different landmark databases shown in Figure 6-10, each of which contains 25 landmarks. As shown in the figure, the maximum-utility database selection approach outperforms the alternative approaches for the majority of the trajectory, providing a more accurate in-flight position estimate. Because these landmark databases are so sparse – each contains only 25 landmarks – jumps can occur in the trajectory estimate due to changes in observation configurations. For example, the maximum-utility position estimation error grows slowly during vehicle launch when only one landmark is observed. However, a second landmark is observed about 22 seconds into the flight, which results in an ambiguous measurement configuration with multiple solutions. The solver selects an incorrect solution, causing the error to jump (recall that our simulated odometry measurements provide weak constraints in comparison to the landmark measurements). A third landmark comes into view and removes the ambiguity almost immediately afterwards. The solver then re-linearizes all of the previous measurements to reflect the new estimate, correcting its previous mistake and extracting more information from these measurements.

Figure 6-11 provides the position estimation errors of Figure 6-9 broken down by axis with their 2-sigma position error covariances, as computed by the graph-based solver. The solver uses the same method to compute the position error covariances in each case, and thus the error covariances reflect the information content of the landmark measurements and enable comparison of the trajectory estimates free from the effects of random measurement noise. The TRN problem is highly nonlinear due to the bearing-only, projective line-of-sight landmark measurements, and therefore the reported covariance is only a first-order approximation (as in an EKF) of the true uncertainty of the estimator. Nonetheless, we can see from the figure that the position error is well-characterized by the covariance estimate of the graph-based solver. For all three axes, the position error covariance when using our maximum-utility database is nearly entirely contained within the boundaries of the errors of the alternative methods, showing that our database selection method results in more precise in-flight position estimates than the alternative landmark selection methods.

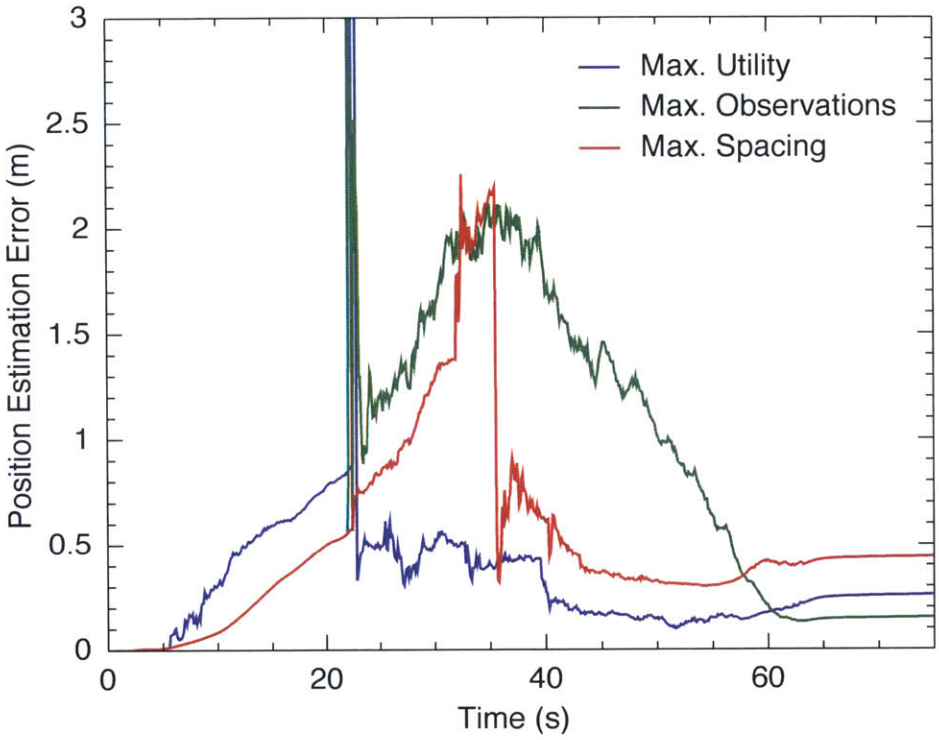
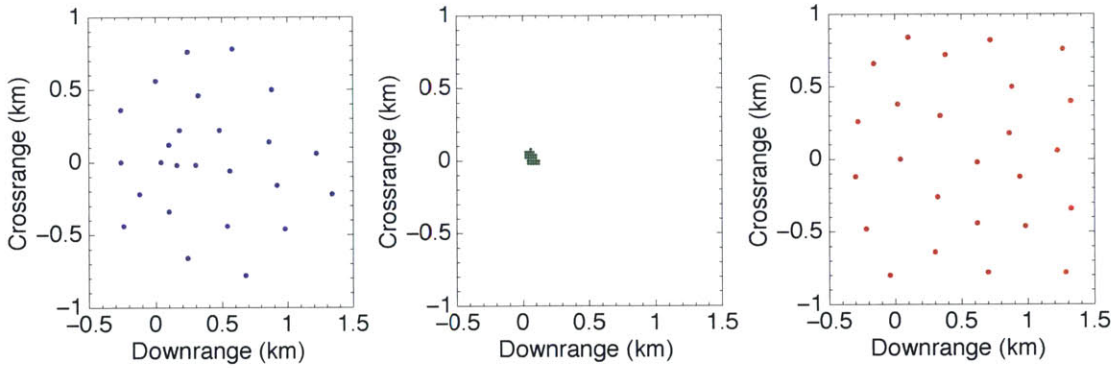


Figure 6-9: Comparison of the total error of the real-time position estimate for the GENIE Campaign 5 flight trajectory when using the three 25-landmark terrain databases shown in Figure 6-10, which were computed using three different selection methods. Especially in the well-constrained approach and landing segments of the trajectory (after the 25 second launch phase), the maximum-utility database results in a lower real-time position estimation error over the majority of the trajectory.



(a) Maximum Utility (b) Maximum Observations (c) Maximum Dispersion

Figure 6-10: Comparison of the 25-landmark databases used in Figure 6-9.

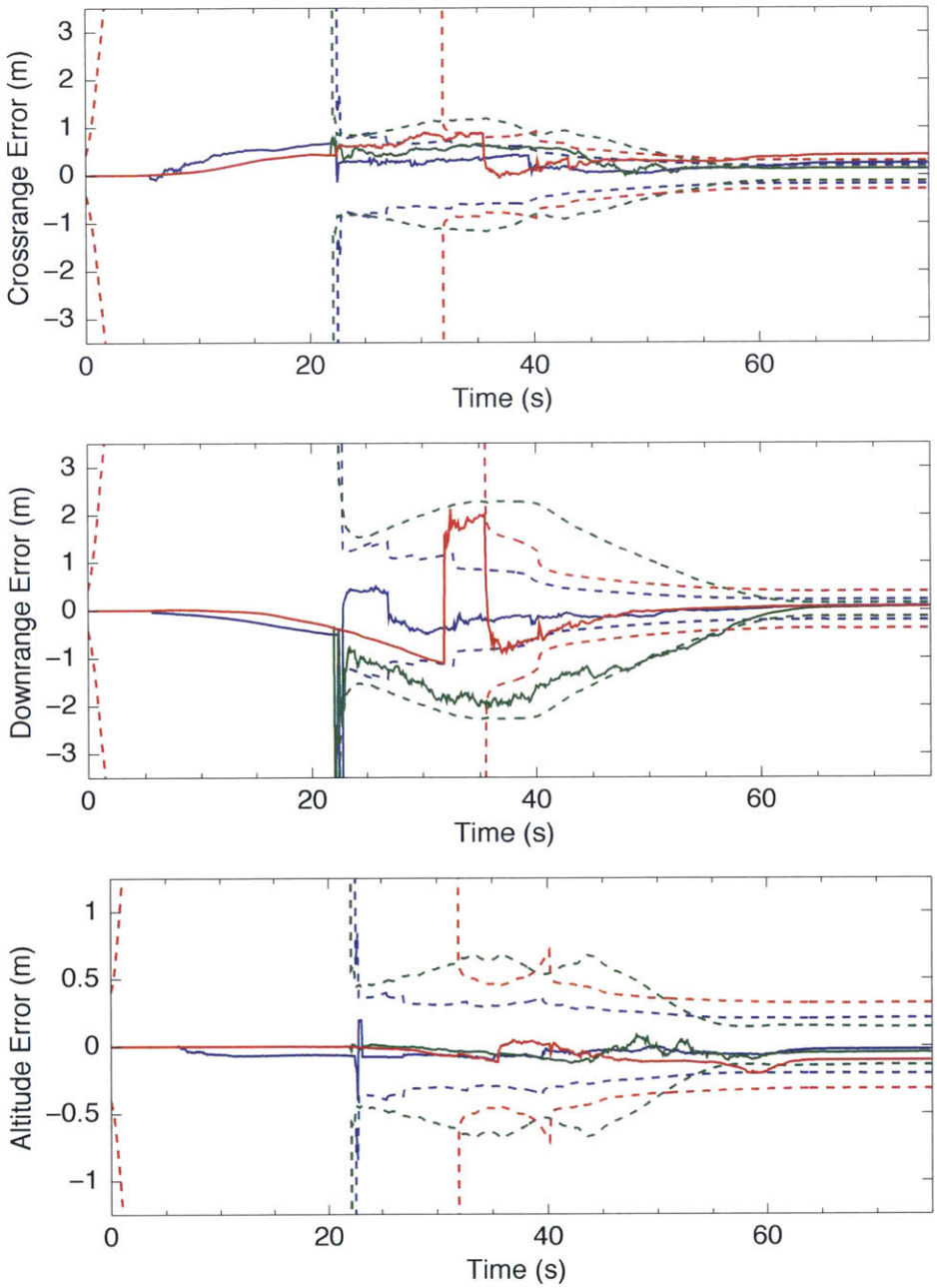


Figure 6-11: Real-time position estimation errors (solid lines) and estimated 2-sigma error bounds (dashed lines) for the simulations presented in Figure 6-9, which compare the results of three selection methods: maximizing landmark spatial distribution (red), maximizing total observations (green), and maximizing landmark utility (blue). The graph-based TRN solver results in an accurate estimate of the position error, and the proposed utility-maximization database selection method's error is nearly entirely contained within those of the alternative methods.

6.6.2 Comparison of Landmark Database Sizes

We now compare trajectory estimates computed using variously-sized maximum-utility landmark databases. Figure 6-12 shows a comparison of the estimator's real-time position error when using terrain databases containing the top 25 and 100 maximum-utility landmarks and a baseline case of 10,201 landmarks spaced evenly across the flight environment on a 20×20 meter grid, as shown in Figure 6-13. Figure 6-14 breaks down these real-time position errors by axis, additionally showing their 2-sigma covariance approximations.

As expected, the trajectory estimation error decreases as database size increases. However, the difference between the trajectory estimates is generally small, especially in the well-constrained approach and landing portions of the trajectory (after the 25 second ignition and launch phase), indicating that even very large landmark databases do not provide significantly more information content than the highly-reduced, 25-landmark database. Furthermore, the error covariances resulting from the 25-landmark case are very close to those of both the 100-landmark database and baseline database, as shown in Figure 6-14. This indicates that utility-based greedy landmark selection succeeds at maximizing the information content of a size-limited landmark database, that the marginal information content of an additional landmark tends to decrease with database size, and that a small but optimal database is nearly as beneficial as the complete database. Thus, increasing the size of the landmark database results in diminishing returns on estimator accuracy and precision.

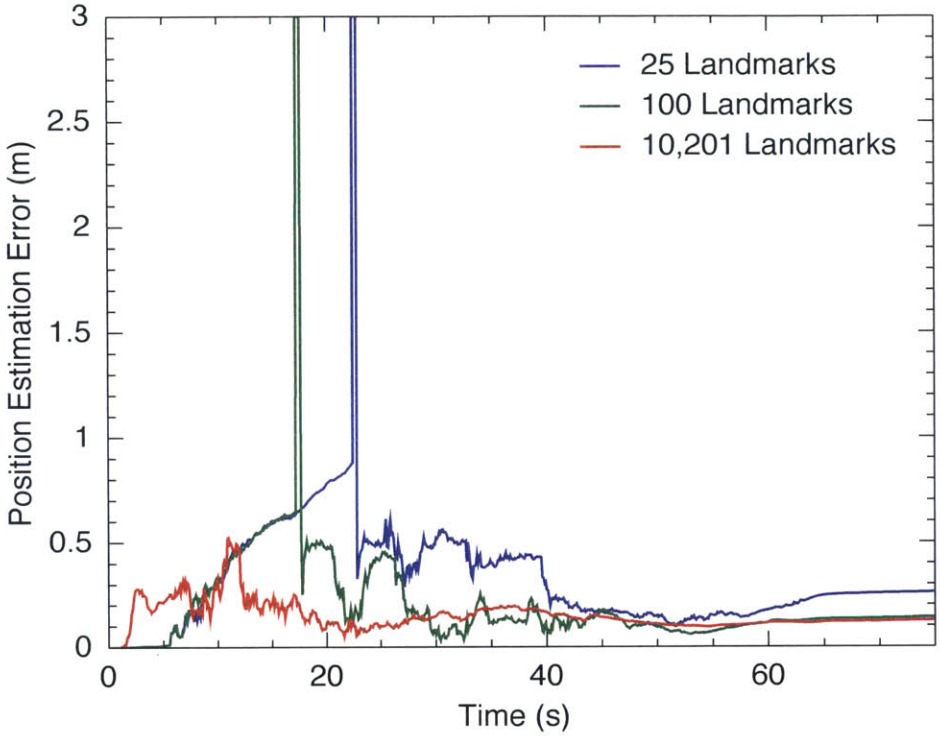
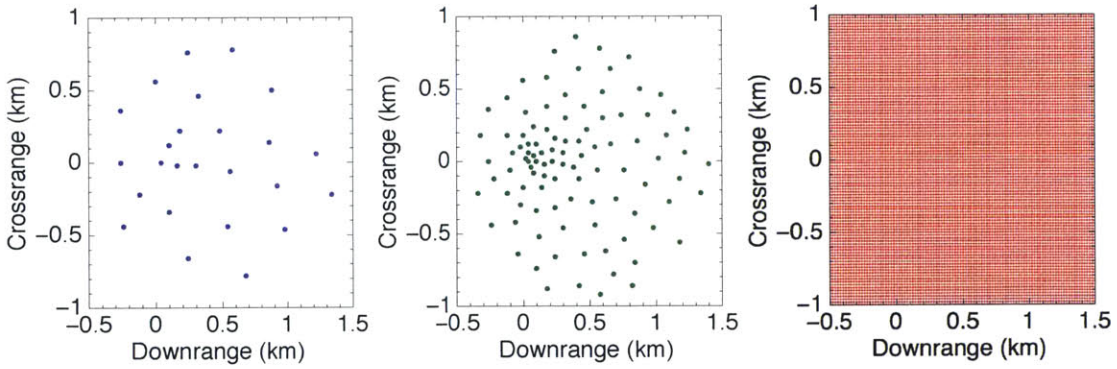


Figure 6-12: Comparison of the total error of the real-time position estimate for the GENIE Campaign 5 flight trajectory when using maximum-utility databases containing 25 and 100 landmarks and a baseline case of a 10,201 landmarks evenly spaced in the flight environment. These landmark databases are shown in Figure 6-13. The 25-landmark database approaches the error of the baseline case despite containing only 0.2% as many landmarks, indicating that even a small number of maximum-utility landmarks contains the majority of the information content of the baseline database.



(a) 25 Landmarks (b) 100 Landmarks (c) 10,201 Landmarks

Figure 6-13: Comparison of the 25 and 100 maximum-utility landmark databases and baseline database of 10,201 landmarks evenly spaced on a 20×20 meter grid used in Figure 6-12.

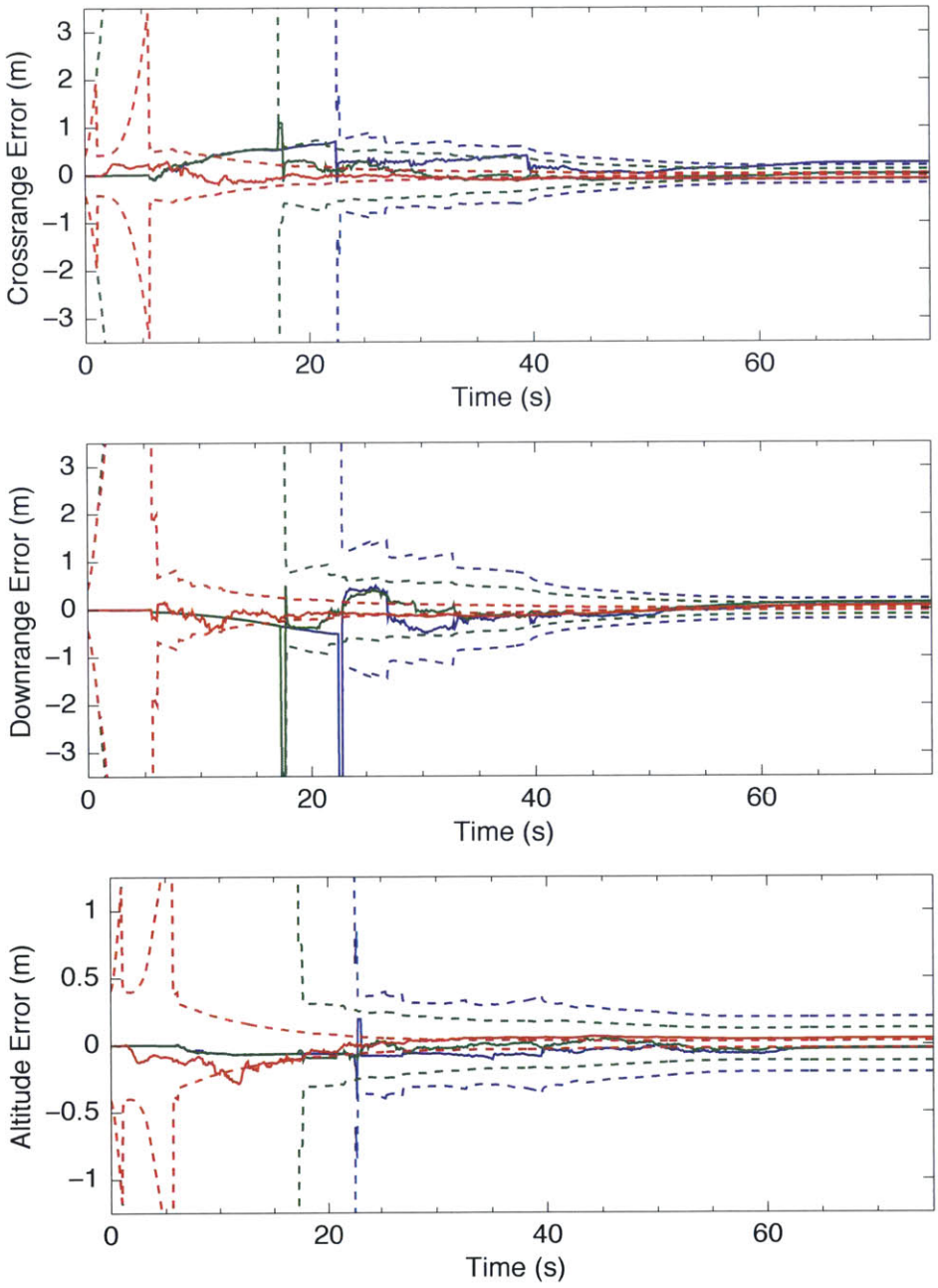


Figure 6-14: Real-time position estimation errors (solid lines) and estimated 2-sigma error bounds (dashed lines) for the simulations presented in Figure 6-12, which compare trajectory estimations using maximum-utility databases of 25 (blue) and 100 (green) landmarks and a baseline case containing 10,201 landmarks (red) evenly spaced on a 20×20 meter grid in the flight region.

6.7 Chapter Summary

This chapter applied the utility-based database selection approach to landmark selection for terrain relative navigation (TRN) in the context of the GENIE and TRNDI systems (described in Section 6.3). Maximum-utility landmark selection enables optimal terrain database construction without direct observation of the vehicle trajectory, as required for TRN. Section 6.4 described how a probabilistic representation of the predicted vehicle trajectory can be computed using Monte Carlo trajectory marginalization, enabling optimal *a priori* terrain database selection. Section 6.5 introduced four metrics of landmark database quality independent of any particular landmark-based TRN solver and showed that the maximum-utility landmark database outperforms those selected randomly or with alternative approaches for the GENIE Campaign 5 flight trajectory.

Section 6.2 introduced a novel graph-based incremental smoothing approach to solve TRN problems. This solver is capable of re-linearizing past measurements as new information is acquired to efficiently achieve the best-possible trajectory estimate. This means the solver can correct past linearization errors, such as those resulting from ambiguous or under-constrained measurement configurations. The solver therefore maximally utilizes the information content of all landmark measurements, making it an ideal choice for reduced-landmark trajectory estimation. Section 6.6 used the graph-based TRN solver and the TRNDI TRN simulator (described in Section 6.3) to show that the maximum-utility landmark database outperforms alternatively selected landmark databases (Section 6.6.1) and that accurate trajectory estimates can be achieved even when using highly reduced landmark databases (Section 6.6.2).

Chapter 7

Conclusions & Future Work

Recently there has been a trend toward data and processing intensive localization and SLAM systems, which aim to estimate a complete representation of a vehicle's environment. However, much of this data is redundant or contains relatively low information content, and unnecessarily stresses the constrained hardware systems on which they run.

This thesis addressed this problem by defining a measure of sensor measurement utility for navigation, which can be computed prior to vehicle operation given a set of basic information that is typically readily available, formulating an optimization problem to maximize the utility of the location database, and providing a greedy approximation algorithm to efficiently construct a location database. Prior to this thesis, the localization and SLAM literature has not addressed location database selection in advance of vehicle navigation and without direct observability of sensor measurements or the vehicle trajectory.

The results presented in this thesis have shown that the locations at which loop-closure events occur (including landmark observations) can be sorted according to their expected reduction in map uncertainty for a general, unknown route, even without access to the pose marginal covariances or actual measurements. In particular, the proposed location utility consists of the visit probability and spatial distribution of database locations.

The results further show that the structure of a vehicle's operational environment and motion can be encoded in the proposed measure of location utility, which in turn improves performance of reduced-size location databases. Using the location utility metric to construct limited-size location databases, an accurate pose-graph can be constructed even when using a location database less than 1% the size of that of a typical pose-graph SLAM system. The greedy-selection algorithm can additionally be used as a design tool to estimate the minimum data storage required to navigate in an environment within a specified position uncertainty.

Chapter 6 applied the proposed measure of landmark utility in the context of terrain relative navigation, showing that location utility can be computed in 3D environments without direct observation of the vehicle trajectory or a finite number of potential vehicle routes. The utility metric was used to optimize a database of landmarks efficiently using the greedy approximation algorithm described in Chapter 4. The presented results showed that, for a fixed-size landmark database, the utility-maximization method results in a higher quality landmark database than random landmark selection or evenly spacing landmarks on a grid for *a priori* database computation, confirming the assertion that not all landmarks are equally valuable for navigation.

In order to validate the landmark selection method, a novel incremental smoothing approach for terrain relative navigation was additionally introduced, which is capable of fusing multiple navigation sensors in an efficient optimization framework. The presented results indicate that this is a worthwhile framework for further TRN study and possible flight testing.

7.1 Recommendations for Future Work

While the core contributions have been demonstrated in a variety of simulations, a detailed experimental demonstration would still be of value, showing that a vehicle can navigate when loop-closures are limited to a finite set of pre-determined locations. However, no existing datasets contain all the characteristics required to perform a comprehensive experimental validation, and thus new datasets will need to be collected. This is primarily due to the lack of structured motion and full environmental descriptions in existing datasets, in which vehicles tend to wander or aim for areal coverage rather than efficient routing, making it difficult to estimate a meaningful viewing probability distribution.

Aside from experimental validation, the database selection approach described in this thesis opens the door to many exciting areas for future research, including online structure learning, tight integration with the place recognition system, and application to multi-agent systems with low-bandwidth communication channels.

7.1.1 Online Structure Learning & Database Management

While this work is limited to *a priori* computation of location databases, which requires vehicle routing and environment information, it opens the door to a vast arena of future work. The most obvious and compelling avenue for future work is that of learning the environment and routing structure online and managing a reduced-size location database in real-time.

Difficult questions arise when managing the database in real-time, such as when to replace established locations with new locations. A number of strategies are possible, but it is likely that the position of the vehicle will factor into decision making. For example, a database managed online might emphasize locations with close proximity to the vehicle, removing distant locations from the database, or it might maintain a global "static" database and a local "active" database.

7.1.2 Place Descriptor Learning

Another pathway for future work involves more tightly integrating the place recognition system into the location database selection. This work currently allows for the expected capability of the place recognition system to recognize various locations to be integrated directly into the location utility determination, as shown in Equation 3.3. However, as discussed in Section 2.3, many appearance-based loop-closure detection systems use information-based methods to compress the sensor data, such as the venerable "Bag of Visual Words" methods. Compression is enabled by learning the distribution of descriptors in the environment, and thus the performance of the place recognition system is highly dependent upon both the number and uniqueness of the locations in the environment being described.

As discussed in Section 2.3.3, FAB-MAP 2.0 achieved 49% recall on a 70 km dataset, but only 3% recall for the 1,000 km dataset. By determining the most valuable loop-closure locations prior to navigation and training the system to recognize these locations especially robustly, we may be able to boost the overall recall of the system on the 1,000 km dataset while also encouraging the place recognition successes to occur at valuable locations.

Bag-of-visual-words matching approaches originated in the object detection and recognition field of computer vision [100]. The PASCAL Challenge [112] led to a great deal of research progress with regards to classifying objects within images into predetermined categories. As discussed in Section 2.3.2, the performance of such object recognition systems was found to be highly dependent upon the quality of the dictionary of visual words, which must be computed during an *a priori* learning phase. Appearance-based place recognition systems for SLAM, including those using bag-of-words approaches, effectively treat each vehicle pose as a category and attempt to classify each new image as either one of these existing categories or a new category. The location selection approach described in this thesis allows valuable *locations* to be determined prior to vehicle navigation, thus potentially enabling better bag-of-words

vocabulary learning approaches for SLAM by allowing physical locations in space to be treated as recognition categories with multiple, designated training images for each location. Thus the objective of vocabulary training shifts from optimally disambiguating images to optimally recognizing specific locations, which may lead to better matching performance and reliability. Disambiguating locations rather than images might also lead to a reduction in the vocabulary sizes required for effective place recognition.

Because place recognition system performance is dependent on the places being recognized, the optimization approach may need to iterate between selecting the database locations and training the place recognition system to achieve the optimal set of locations. This is an especially exciting area for future research that has thus far been unexplored, and served as the initial motivation for the work presented in this thesis.

7.1.3 Multi-Agent Localization & Mapping

The pose-graphs generated by the proposed research typically involve long segments of poses connected solely by odometry constraints with occasional loop-closure constraints. Because we know that loop-closure constraints cannot be added to these poses (any poses that are definitely not in the location database), they are generally safe to marginalize out, effectively summarizing large portions of the pose-graph. The result will be a series of constraints between only the poses in the location database. Thus, these summarized constraints can easily be shared with other vehicles across a network. As these constraints can be applied in any order and at any time, regardless of whether other agents have visited the location yet or not, this significantly reduces challenges associated with measurement ordering and low-bandwidth, high-latency communication channels.

7.1.4 Terrain Relative Navigation

While terrain relative navigation for planetary landing is an active area of recent research, there is a significant lack of experimental validation of algorithms in relevant environments. The TRNDI System, described in Section 6.3.2, was originally designed and assembled to provide such a demonstration onboard a terrestrial rocket flight. We continue to hope for an opportunity for a flight demonstration of this system.

An additional pathway for future work on the TRN system includes extending the calculation of the location viewing probability distribution to account for vehicle altitude, allowing database selection accounting for large vehicle altitude changes, such as occur during planetary landing. While the presented TRN solver is capable of seamlessly handling multi-resolution landmark databases, optimal landmark selection across a continuous resolution space remains an open question, likely requiring a modification to the spatial dispersion term in the utility computation to account for the increased dimensionality of the selection space. Existing systems use multiple landmark databases computed independently for each resolution level.

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics.* MIT press, 2005.

[2] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics & Automation Magazine* 13.2 (2006), pp. 99–110.

[3] G. Grisetti et al. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[4] V. Ila, J. M. Porta, and J. Andrade-Cetto. "Information-based compact Pose SLAM". In: *IEEE Transactions on Robotics* 26.1 (2010), pp. 78–93.

[5] K. Lynch. *The image of the city.* Vol. 11. MIT press, 1960.

[6] L. Vincent. "Taking online maps down to street level". In: *Computer* 40.12 (2007), pp. 118–20.

[7] S. Thrun. "Robotic mapping: A survey". In: *Exploring artificial intelligence in the new millennium* (2002).

[8] P Cheeseman, R Smith, and M Self. "A stochastic map for uncertain spatial relationships". In: *4th International Symposium on Robotic Research.* 1987, pp. 467–74.

[9] J. Leonard and H. Durrant-Whyte. "Simultaneous map building and localization for an autonomous mobile robot". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* Vol. 3. 1991, pp. 1442–7.

[10] M. G. Dissanayake et al. "A solution to the simultaneous localization and map building (SLAM) problem". In: *IEEE Transactions on Robotics and Automation* 17.3 (2001), pp. 229–241.

[11] B. Kuipers and Y.-T. Byun. "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations". In: *Robotics and autonomous systems* 8.1 (1991), pp. 47–63.

[12] H. Choset and K. Nagatani. "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization". In: *IEEE Transactions on Robotics and Automation* 17.2 (2001), pp. 125–37.

[13] M. Cummins and P. Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance". In: *The International Journal of Robotics Research* 27.6 (2008), pp. 647–65.

[14] B. Kuipers. "The spatial semantic hierarchy". In: *Artificial Intelligence* 119.1 (2000), pp. 191–233.

[15] B. Kuipers. *The map-learning critter.* Tech. rep. University of Texas at Austin, 1985.

[16] M. J. Milford, G. F. Wyeth, and D. Prasser. "RatSLAM: a hippocampal model for simultaneous localization and mapping". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* Vol. 1. 2004, pp. 403–8.

[17] K. Konolige, E. Marder-Eppstein, and B. Marthi. "Navigation in hybrid metric-topological maps". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2011, pp. 3041–3047.

[18] S. Tully, G. Kantor, and H. Choset. "A unified Bayesian framework for global localization and SLAM in hybrid metric/topological maps". In: *The International Journal of Robotics Research* 31.3 (2012), pp. 271–88.

[19] P. Beeson, J. Modayil, and B. Kuipers. "Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarachy". In: *The International Journal of Robotics Research* 29.4 (2009), pp. 428–59.

[20] J.-L. Blanco, J.-A. Fernández-Madrigal, and J. Gonzalez. "Toward a Unified Bayesian Approach to Hybrid Metric–Topological SLAM". In: *IEEE Transactions on Robotics* 24.2 (2008), pp. 259–70.

[21] T. Bailey and H. Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II". In: *IEEE Robotics & Automation Magazine* 13.3 (2006), pp. 108–17.

[22] R. Smith, M. Self, and P. Cheeseman. "Estimating uncertain spatial relationships in robotics". In: *Autonomous robot vehicles.* 1990, pp. 167–93.

[23] A. Gelb. *Applied optimal estimation.* MIT Press, 1974.

[24] P. Moutarlier and R. Chatila. "An experimental system for incremental environment modelling by an autonomous mobile robot". In: *International Symposium on Experimental Robotics.* 1990, pp. 327–46.

[25] A. J. Davison and D. W. Murray. "Simultaneous localization and map-building using active vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (2002), pp. 865–880.

[26] A. J. Davison. "Real-time simultaneous localisation and mapping with a single camera". In: *Proceedings of the 9th IEEE International Conference on Computer Vision.* 2003, pp. 1403–10.

[27] A. J. Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 1052–67.

[28] M. Bosse et al. "An atlas framework for scalable mapping". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* Vol. 2. 2003, pp. 1899–906.

[29] C. Estrada, J. Neira, and J. D. Tardós. "Hierarchical SLAM: Real-time accurate mapping of large environments". In: *IEEE Transactions on Robotics* 21.4 (2005), pp. 588–96.

[30] M. Li and A. I. Mourikis. "High-precision, consistent EKF-based visual-inertial odometry". In: *The International Journal of Robotics Research* 32.6 (2013), pp. 690–711.

[31] J. Civera, A. J. Davison, and J Montiel. "Inverse depth parametrization for monocular SLAM". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 932–45.

[32] M. Montemerlo et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2002, pp. 593–8.

[33] M. Montemerlo and S. Thrun. "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges". In: *IEEE Transactions on Robotics* (2003).

[34] E. Eade and T. Drummond. "Scalable monocular SLAM". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1. 2006, pp. 469–76.

[35] S. Thrun et al. "Simultaneous localization and mapping with sparse extended information filters". In: *The International Journal of Robotics Research* 23.7-8 (2004), pp. 693–716.

[36] R. Eustice, H. Singh, and J. Leonard. "Exactly Sparse Delayed-State Filters for View-Based SLAM". In: *IEEE Transactions on Robotics* 22.6 (2006), pp. 1100–14.

[37] M. R. Walter, R. M. Eustice, and J. J. Leonard. "Exactly sparse extended information filters for feature-based SLAM". In: *The International Journal of Robotics Research* 26.4 (2007), pp. 335–59.

[38] D. Gamage and T. Drummond. "Reduced Dimensionality Extended Kalman Filter for SLAM". In: *Proceedings of the British Machine Vision Conference*. 2013.

[39] S. Koenig and R. G. Simmons. "Unsupervised learning of probabilistic models for robot navigation". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 3. 1996, pp. 2301–8.

[40] H. Shatkay and L. P. Kaelbling. "Learning topological maps with weak local odometric information". In: *Proceedings of the International Joint Conferences on Artificial Intelligence*. 1997, pp. 920–9.

[41] S. Thrun, W. Burgard, and D. Fox. "A probabilistic approach to concurrent mapping and localization for mobile robots". In: *Autonomous Robots* 5.3-4 (1998), pp. 253–71.

[42] B. Triggs et al. "Bundle adjustment: a modern synthesis". In: *Vision algorithms: theory and practice*. 2000, pp. 298–372.

[43] S. Granshaw. "Bundle adjustment methods in engineering photogrammetry". In: *The Photogrammetric Record* 10.56 (1980), pp. 181–207.

[44] C. Engels, H. Stewénius, and D. Nistér. "Bundle adjustment rules". In: *Photogrammetric Computer Vision* 2 (2006).

[45] G. Sibley. *Relative bundle adjustment*. Tech. rep. 2307. Department of Engineering Science, Oxford University, 2009.

[46] Y. Jeong et al. "Pushing the envelope of modern methods for bundle adjustment". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.8 (2012), pp. 1605–17.

[47] S. Ullman. "The interpretation of structure from motion". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979), pp. 405–26.

[48] R. C. Bolles, H. H. Baker, and D. H. Marimont. "Epipolar-plane image analysis: An approach to determining structure from motion". In: *International Journal of Computer Vision* 1.1 (1987), pp. 7–55.

[49] J. J. Koenderink and A. J. van Doorn. "Affine structure from motion". In: *Journal of the Optical Society of America A* 8.2 (1991), pp. 377–85.

[50] P. Sturm and B. Triggs. "A factorization based algorithm for multi-image projective structure and motion". In: *Proceedings on the European Conference on Computer Vision*. 1996, pp. 709–720.

[51] J. Oliensis. "A critique of structure-from-motion algorithms". In: *Computer Vision and Image Understanding* 80.2 (2000), pp. 172–214.

[52] G. Klein and D. Murray. "Parallel tracking and mapping for small AR workspaces". In: *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. 2007, pp. 225–34.

[53] K. Konolige and M. Agrawal. "FrameSLAM: From bundle adjustment to real-time visual mapping". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1066–77.

[54] N. Snavely, S. M. Seitz, and R. Szeliski. "Skeletal graphs for efficient structure from motion." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2008.

[55] C. Mei et al. "RSLAM: A System for Large-Scale Mapping in Constant-Time Using Stereo". In: *International Journal of Computer Vision* 94.2 (2011), pp. 198–214.

[56] H. Strasdat, J. Montiel, and A. J. Davison. "Scale Drift-Aware Large Scale Monocular SLAM". In: *Robotics: Science and Systems*. Vol. 2. 3. 2010, p. 5.

[57] H. Strasdat, J. Montiel, and A. J. Davison. "Real-time monocular SLAM: Why filter?" In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2010, pp. 2657–64.

[58] M. Golfarelli, D. Maio, and S. Rizzi. "Elastic correction of dead-reckoning errors in map building". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 2. 1998, pp. 905–11.

[59] F. Lu and E. Milios. "Globally consistent range scan alignment for environment mapping". In: *Autonomous robots* 4.4 (1997), pp. 333–49.

[60] S. Thrun and M. Montemerlo. "The Graph SLAM algorithm with applications to large-scale mapping of urban structures". In: *International Journal of Robotics Research* 25.5-6 (2006), pp. 403–29.

[61] G. Sibley et al. "Vast-scale outdoor navigation using adaptive relative bundle adjustment". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 958–80.

[62] H. Strasdat et al. "Double window optimisation for constant time visual SLAM". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2011, pp. 2352–9.

[63] V. Indelman et al. "Information fusion in navigation systems via factor graph based incremental smoothing". In: *Robotics and Autonomous Systems* 61.8 (2013), pp. 721–38.

[64] A. Walcott-Bryant et al. "Dynamic pose graph SLAM: Long-term mapping in low dynamic environments". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2012, pp. 1871–8.

[65] D. Rosen, M. Kaess, and J. Leonard. "RISE: An Incremental Trust-Region Method for Robust Online Sparse Least-Squares Estimation". In: *Robotics, IEEE Transactions on* PP.99 (2014).

[66] R. Kuemmerle et al. "g2o: A general framework for graph optimization". In: *Proceedings of the 2011 IEEE International Conference on Robotics and Automation.* 2011.

[67] M. Kaess, A. Ranganathan, and F. Dellaert. "iSAM: Incremental smoothing and mapping". In: *IEEE Transactions on Robotics* 24.6 (2008), pp. 1365–78.

[68] M. Kaess et al. "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering". In: *Proceedings of the 2011 IEEE International Conference on Robotics and Automation.* 2011, pp. 3281–8.

[69] M. Kaess et al. "Concurrent filtering and smoothing". In: *Proceedings of the 15th International Conference on Information Fusion.* 2012, pp. 1300–7.

[70] K. Konolige, M. Agrawal, and J. Sola. "Large-scale visual odometry for rough terrain". In: *Robotics Research.* 2011, pp. 201–12.

[71] D. Scaramuzza and F. Fraundorfer. "Visual odometry tutorial part I: The First 30 Years and Fundamentals". In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 80–92.

[72] T. Oskiper et al. "Visual Odometry System Using Multiple Stereo Cameras and Inertial Measurement Unit". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2007, pp. 1–8.

[73] E. S. Jones and S. Soatto. "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach". In: *The International Journal of Robotics Research* 30.4 (2011), pp. 407–30.

[74] T. J. Steiner et al. "Unifying Inertial and Relative Solutions for Planetary Surface Navigation". In: *Proceedings of the IEEE Aerospace Conference.* Big Sky, Montana, 2012.

[75] D. Nistér, O. Naroditsky, and J. Bergen. "Visual odometry". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Vol. 1. 2004.

[76] D. Nistér. "An efficient solution to the five-point relative pose problem". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.6 (2004), pp. 756–70.

[77] H. Stewénius, C. Engels, and D. Nistér. "Recent developments on direct relative orientation". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 60.4 (2006), pp. 284–94.

[78] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–95.

[79] M. Maimone, Y. Cheng, and L. Matthies. "Two years of visual odometry on the mars exploration rovers". In: *Journal of Field Robotics* 24.3 (2007), pp. 169–86.

[80] J.-P. Tardif, Y. Pavlidis, and K. Daniilidis. "Monocular visual odometry in urban environments using an omnidirectional camera". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2008, pp. 2531–8.

[81] G. Sibley, L. Matthies, and G. Sukhatme. "Sliding window filter with application to planetary landing". In: *Journal of Field Robotics* 27.5 (2010), pp. 587–608.

[82] J. A. Hesch et al. "Towards consistent vision-aided inertial navigation". In: *Algorithmic Foundations of Robotics X.* 2013, pp. 559–74.

[83] C. Harris and M. Stephens. "A combined corner and edge detector." In: *Alvey vision conference.* Vol. 15. 1988, p. 50.

[84] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[85] H. Bay, T. Tuytelaars, and L. Van Gool. "SURF: Speeded up robust features". In: *European Conference on Computer Vision.* 2006, pp. 404–17.

[86] E. Rosten and T. Drummond. "Machine learning for high-speed corner detection". In: *European Conference on Computer Vision.* 2006, pp. 430–43.

[87] A. Alahi, R. Ortiz, and P. Vandergheynst. "FREAK: Fast retina keypoint". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2012, pp. 510–7.

[88] S. Leutenegger, M. Chli, and R. Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2011, pp. 2548–55.

[89] J.-M. Morel and G. Yu. "ASIFT: A new framework for fully affine invariant image comparison". In: *SIAM Journal on Imaging Sciences* 2.2 (2009), pp. 438–69.

[90] P. F. Alcantarilla, L. M. Bergasa, and A. J. Davison. "Gauge-SURF descriptors". In: *Image and Vision Computing* 31.1 (2013), pp. 103–16.

[91] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. "KAZE features". In: *European Conference on Computer Vision.* 2012, pp. 214–27.

[92] N. Snavely, S. M. Seitz, and R. Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM Transactions on Graphics* 25.3 (2006), pp. 835–46.

[93] N. Snavely, S. M. Seitz, and R. Szeliski. "Modeling the world from internet photo collections". In: *International Journal of Computer Vision* 80.2 (2008), pp. 189–210.

[94] S. Agarwal et al. "Building Rome in a day". In: *Communications of the ACM* 54.10 (2011), pp. 105–12.

[95] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[96] R. I. Hartley. "In defense of the eight-point algorithm". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.6 (1997), pp. 580–93.

[97] J. S. Beis and D. G. Lowe. "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1997, pp. 1000–6.

[98] S. Brin and L. Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer networks and ISDN systems* 30.1 (1998), pp. 107–17.

[99] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval.* ACM Press New York, 1999, p. 463.

[100] J. Sivic and A. Zisserman. "Video Google: A text retrieval approach to object matching in videos". In: *Proceedings of the IEEE International Conference on Computer Vision.* 2003, pp. 1470–7.

[101] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images.* Morgan Kaufmann, 1999.

[102] D. Nistér and H. Stewenius. "Scalable recognition with a vocabulary tree". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* Vol. 2. 2006, pp. 2161–8.

[103] B. Girod et al. "Mobile visual search". In: *IEEE Signal Processing Magazine* 28.4 (2011), pp. 61–76.

[104] E. Eade and T. Drummond. "Unified Loop Closing and Recovery for Real Time Monocular SLAM". In: *Proceedings of the British Machine Vision Conference.* Vol. 13. 2008, p. 136.

[105] G. S. Christopher Mei and P. Newman. "Closing Loops Without Places". In: *Proceedings of the International Conference on Intelligent Robots and Systems.* Taipei, Taiwan, 2010.

[106] D. Galvez-Lopez and J. D. Tardos. "Real-time loop detection with bags of binary words". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2011, pp. 51–8.

[107] K. Konolige et al. "View-based maps". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 941–57.

[108] A. J. Glover et al. "FAB-MAP + RatSLAM: appearance-based SLAM for multiple times of day". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2010, pp. 3507–3512.

[109] W. Churchill and P. Newman. "Practice Makes Perfect? Managing and Leveraging Visual Experiences for Lifelong Navigation". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* St. Paul, MN, 2012.

[110] W. Churchill and P. Newman. "Continually Improving Large Scale Long Term Visual Navigation of a Vehicle in Dynamic Urban Environments". In: *Proceedings of the IEEE Intelligent Transportation Systems Conference.* Anchorage, AK, 2012.

[111] W. Churchill and P. Newman. "Experience-based navigation for long-term localisation". In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1645–61.

[112] O. Chum and A. Zisserman. "An exemplar model for learning object classes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2007.

[113] M.-E. Nilsback and A. Zisserman. "A visual vocabulary for flower classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE Computer Society, 2006, pp. 1447–54.

[114] M. Cummins and P. Newman. "Appearance-only SLAM at Large Scale with FAB-MAP 2.0". In: *The International Journal of Robotics Research* 30.9 (2010), pp. 1100–23.

[115] A. Glover et al. "OpenFABMAP: An open source toolbox for appearance-based loop closure detection". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2012, pp. 4730–4735.

[116] P. Newman et al. "Navigating, Recognising and Describing Urban Spaces With Vision and Laser". In: *The International Journal of Robotics Research* 28 (2009).

[117] P. Piniés et al. "CI-Graph simultaneous localization and mapping for three-dimensional reconstruction of large and complex environments using a multi-camera system". In: *Journal of Field Robotics* 27.5 (2010), pp. 561–86.

[118] W. Maddern, M. Milford, and G. Wyeth. "CAT-SLAM: probabilistic localisation and mapping using a continuous appearance-based trajectory". In: *The International Journal of Robotics Research* 31.4 (2012), pp. 429–51.

[119] J. Vial, H. Durrant-Whyte, and T. Bailey. "Conservative sparsification for efficient and consistent approximate estimation". In: *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 886–93.

[120] M. Chli and A. J. Davison. "Automatically and Efficiently Inferring the Hierarchical Structure of Visual Maps". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2009, pp. 387–94.

[121] R. O. Castle, G. Klein, and D. W. Murray. "Wide-area augmented reality using camera tracking and mapping in multiple regions". In: *Computer Vision and Image Understanding* 115.6 (2011), pp. 854–67.

[122] K. Pirker, M Ruther, and H. Bischof. "CD SLAM-continuous localization and mapping in a dynamic world". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 3990–7.

[123] K. Ni, D. Steedly, and F. Dellaert. "Tectonic SAM: Exact, out-of-core, submap-based SLAM". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2007, pp. 1678–85.

[124] S. Guha and S. Khuller. "Approximation algorithms for connected dominating sets". In: *Algorithmica* 20.4 (1998), pp. 374–87.

[125] O Booij, Z Zivkovic, and B Kröse. "Pruning the image set for appearance based robot localization". In: *Proceedings of the Annual Conference of the Advanced School for Computing and Imaging*. 2005.

[126] O. Booij, Z. Zivkovic, and B Kröse. "Efficient data association for view based SLAM using connected dominating sets". In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1225–34.

[127] E. Eade, P. Fong, and M. E. Munich. "Monocular graph SLAM with complexity reduction". In: *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 3017–24.

[128] N. Carlevaris-Bianco and R. M. Eustice. "Generic factor-based node marginalization and edge sparsification for pose-graph SLAM". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 5748–55.

[129] N. Carlevaris-Bianco and R. M. Eustice. "Long-term simultaneous localization and mapping with generic linear constraint node removal". In: *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 1034–41.

[130] N. Carlevaris-Bianco, M. Kaess, and R. M. Eustice. "Generic Node Removal for Factor-Graph SLAM". In: *IEEE Transactions on Robotics* (2014).

[131] N. Carlevaris-Bianco and R. M. Eustice. "Conservative edge sparsification for graph SLAM node removal". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Hong Kong, China, 2014, pp. 854–860.

[132] G. Huang, M. Kaess, and J. J. Leonard. "Consistent sparsification for graph optimization". In: *Proceedings of the 2013 European Conference on Mobile Robots*. 2013, pp. 150–7.

[133] A. Kim and R. M. Eustice. "Combined visually and geometrically informative link hypothesis for pose-graph visual SLAM using bag-of-words". In: *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 1647–54.

[134] P. Ozog and R. M. Eustice. "Toward long-term, automated ship hull inspection with visual SLAM, explicit surface optimization, and generic graph-sparsification". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Hong Kong, China, 2014, pp. 3832–9.

[135] C. E. Shannon. "A mathematical theory of communication". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), pp. 3–55.

[136] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[137] M. Chli. "Applying information theory to efficient SLAM". PhD thesis. Imperial College London, 2010.

[138] H. Kretzschmar, C. Stachniss, and G. Grisetti. "Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders". In: *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 865–71.

[139] H. Kretzschmar and C. Stachniss. "Information-theoretic compression of pose graphs for laser-based SLAM". In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1219–30.

[140] Y. Latif and J. Neira. "Go straight, turn right: Pose graph reduction through trajectory segmentation using line segments". In: *Proceedings of the European Conference on Mobile Robots*. 2013, pp. 144–9.

[141] Y. Wang et al. "Kullback-Leibler divergence based graph pruning in robotic feature mapping". In: *Proceedings of the 2013 European Conference on Mobile Robots*. 2013, pp. 32–7.

[142] A. Kim and R. M. Eustice. "Perception-driven navigation: Active visual SLAM for robotic area coverage". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany, 2013, pp. 3181–3188.

[143] A. Kim and R. M. Eustice. "Active visual SLAM for robotic area coverage: Theory and experiment". In: *International Journal of Robotics Research* 34.4-5 (2015), pp. 457–75.

[144] M. J. Milford and G. F. Wyeth. "Mapping a suburb with a single camera using a biologically inspired SLAM system". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1038–53.

[145] M. Milford and G. Wyeth. "Persistent navigation and mapping using a biologically inspired SLAM system". In: *The International Journal of Robotics Research* 29.9 (2010), pp. 1131–53.

[146] H. Johannsson et al. "Temporally scalable visual SLAM using a reduced pose graph". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 54–61.

[147] M. Labbe and F. Michaud. "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation". In: *IEEE Transactions on Robotics* 29.3 (2013), pp. 734–45.

[148] A. Krause, A. Singh, and C. Guestrin. "Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies". In: *The Journal of Machine Learning Research* 9 (2008), pp. 235–84.

[149] M. Beinhofer, J. Müller, and W. Burgard. "Effective landmark placement for accurate and reliable mobile robot navigation". In: *Robotics and Autonomous Systems* 61.10 (2013), pp. 1060–9.

[150] R. Allen et al. "The Range Beacon Placement Problem for Robot Navigation". In: *2014 Canadian Conference on Computer and Robot Vision*. 2014, pp. 151–8.

[151] M. P. Vitus and C. J. Tomlin. "Sensor Placement for Improved Robotic Navigation." In: *Robotics: Science and Systems*. 2010.

[152] S. Frintrop and P. Jensfelt. "Attentional landmarks and active gaze control for visual SLAM". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1054–65.

[153] S. Hochdorfer and C. Schlegel. "Landmark rating and selection according to localization coverage: Addressing the challenge of lifelong operation of SLAM in service robots". In: *Proceedings of the IEEE/RSJ international conference on Intelligent robots and systems*. IEEE Press. 2009, pp. 382–7.

[154] S. Zhang, L. Xie, and M. D. Adams. "Entropy based feature selection scheme for real time simultaneous localization and map building". In: *Proceedings of the IEEE/RSJ international conference on Intelligent robots and systems*. IEEE. 2005, pp. 1175–80.

[155] G. Dissanayake, H. Durrant-Whyte, and T. Bailey. "A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* Vol. 2. IEEE. 2000, pp. 1009–14.

[156] H. Strasdat, C. Stachniss, and W. Burgard. "Which landmark is useful? Learning selection policies for navigation in unknown environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* IEEE. 2009, pp. 1410–5.

[157] J. A. Bondy and U. S. R. Murty. *Graph theory with applications.* Vol. 6. Macmillan London, 1976.

[158] T. H. Cormen et al. *Introduction to algorithms.* Vol. 2. MIT Press Cambridge, 2001.

[159] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* (1959), pp. 269–71.

[160] M. L. Fredman and R. E. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *Journal of the ACM* 34.3 (1987), pp. 596–615.

[161] R. K. Ahuja et al. "Faster algorithms for the shortest path problem". In: *Journal of the ACM* 37.2 (1990), pp. 213–23.

[162] P. E. Hart, N. J. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–7.

[163] R. W. Floyd. "Algorithm 97: shortest path". In: *Communications of the ACM* 5.6 (1962), p. 345.

[164] D. B. Johnson. "Efficient algorithms for shortest paths in sparse networks". In: *Journal of the ACM (JACM)* 24.1 (1977), pp. 1–13.

[165] L. E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–80.

[166] S. M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Tech. rep. Computer Science Department, Iowa State University, 1998.

[167] J. J. Kuffner and S. M. LaValle. "RRT-Connect: An efficient approach to single-query path planning". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* Vol. 2. 2000, pp. 995–1001.

[168] S. Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of Field Robotics* 23.9 (2006), pp. 661–92.

[169] J. Leonard et al. *Team MIT Urban Challenge technical report.* Tech. rep. Massachusetts Institute of Technology, 2007.

[170] J. Poppinga et al. "Fast 6-DOF path planning for autonomous underwater vehicles (AUV) based on 3D plane mapping". In: *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics.* 2011, pp. 345–50.

[171] J.-A. Meyer and D. Filliat. "Map-based navigation in mobile robots II: a review of map-learning and path-planning strategies". In: *Cognitive Systems Research* 4.4 (2003), pp. 283–317.

[172] S. Shen, N. Michael, and V. Kumar. "Autonomous multi-floor indoor navigation with a computationally constrained MAV". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2011, pp. 20–5.

[173] F. Fraundorfer et al. "Vision-based autonomous mapping and exploration using a quadrotor MAV". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2012, pp. 4557–4564.

[174] F. S. Hover et al. "Advanced perception, navigation and planning for autonomous in-water ship hull inspection". In: *The International Journal of Robotics Research* 31.12 (2012), pp. 1445–64.

[175] R. Kummerle et al. "A navigation system for robots operating in crowded urban environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2013, pp. 3225–32.

[176] M. Whitty and J. Guivant. "Efficient global path planning during dense map deformation". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2011, pp. 4943–9.

[177] S. Prentice and N. Roy. "The belief roadmap: Efficient planning in belief space by factoring the covariance". In: *The International Journal of Robotics Research* 28.11-12 (2009), pp. 1448–65.

[178] R. He, S. Prentice, and N. Roy. "Planning in information space for a quadrotor helicopter in a GPS-denied environment". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2008, pp. 1814–20.

[179] R. Valencia, J. Andrade-Cetto, and J. M. Porta. "Path planning in belief space with Pose SLAM". In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2011, pp. 78–83.

[180] R. Valencia et al. "Planning reliable paths with Pose SLAM". In: *IEEE Transactions on Robotics* 29.4 (2013), pp. 1050–9.

[181] R. Valencia et al. "Active pose SLAM". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2012, pp. 1885–91.

[182] H. Carrillo et al. "Fast minimum uncertainty search on a graph map representation". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2012, pp. 2504–11.

[183] M. Juliá, A. Gil, and O. Reinoso. "A comparison of path planning strategies for autonomous exploration and mapping of unknown environments". In: *Autonomous Robots* 33.4 (2012), pp. 427–44.

[184] B. Stenning and T. D. Barfoot. "Path planning on a network of paths". In: *Proceedings of the IEEE Aerospace Conference.* 2011.

[185] B Stenning et al. "Planetary surface exploration using a network of reusable paths". In: *Proceedings of the 43rd Lunar and Planetary Science Conference.* 2012.

[186] M. Kaess and F. Dellaert. "Covariance recovery from a square root information matrix for data association". In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1198–210.

[187] L. C. Freeman. "A set of measures of centrality based on betweenness". In: *Sociometry* (1977), pp. 35–41.

[188] D. J. White. "The maximal dispersion problem and the "first point outside the neighbourhood" heuristic". In: *Computers & Operations Research* 18.1 (1991), pp. 43–50.

[189] E. Erkut. "The discrete p-dispersion problem". In: *European Journal of Operational Research* 46.1 (1990), pp. 48–60.

[190] E. Erkut, Y. Ülküsal, and O. Yenicerioğlu. "A comparison of p-dispersion heuristics". In: *Computers & operations research* 21.10 (1994), pp. 1103–13.

[191] E. Erkut and S. Neuman. "Analytical models for locating undesirable facilities". In: *European Journal of Operational Research* 40.3 (1989), pp. 275–91.

[192] B. Chandra and M. M. Halldórsson. "Approximation algorithms for dispersion problems". In: *Journal of Algorithms* 38.2 (2001), pp. 438–65.

[193] R. M. Karp. "Reducibility among combinatorial problems". In: *Complexity of Computer Calculations*. Ed. by R. E. Miller and J. W. Thatcher. New York: Plenum, 1972, pp. 85–103.

[194] A. Dasgupta, R. Kumar, and S. Ravi. "Summarization Through Submodularity and Dispersion". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. 2013, pp. 1014–22.

[195] A. Tamir. "Obnoxious facility location on graphs". In: *SIAM Journal on Discrete Mathematics* 4.4 (1991), pp. 550–67.

[196] M. I. Shamos. "Computational geometry." PhD thesis. 1978.

[197] G. T. Toussaint. "Computing largest empty circles with location constraints". In: *International journal of computer & information sciences* 12.5 (1983), pp. 347–58.

[198] R. Z. Farahani, M. SteadieSeifi, and N. Asgari. "Multiple criteria facility location problems: A survey". In: *Applied Mathematical Modelling* 34.7 (2010), pp. 1689–1709.

[199] J. Kenyon and M. Law. *Method and apparatus for distributing and displaying maps electronically*. 2004.

[200] J. Sacks. *Techniques for displaying and caching tiled map data on constrained-resource services*. 2008.

[201] M. Haklay and P. Weber. "OpenStreetMap: User-generated street maps". In: *Pervasive Computing* 7.4 (2008), pp. 12–18.

[202] T. Opsahl, F. Agneessens, and J. Skvoretz. "Node centrality in weighted networks: Generalizing degree and shortest paths". In: *Social Networks* 32.3 (2010), pp. 245–51.

[203] T. J. Steiner, G. Huang, and J. J. Leonard. "Location Utility-based Map Reduction". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Seattle, Washington, 2015.

[204] J. Bezanson et al. "Julia: A Fast Dynamic Language for Technical Computing". In: *arXiv Computing Research Repository* abs/1209.5145 (2012).

[205] W. Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32.5 (1976), pp. 922–3.

[206] W. Kabsch. "A discussion of the solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 34.5 (1978), pp. 827–8.

[207] T. J. Steiner and T. M. Brady. "Vision-Based Navigation and Hazard Detection for Terrestrial Rocket Approach and Landing". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2014.

[208] T. Brady, T. Crain, and S. Paschall. "ALHAT System Validation". In: *Proceedings of the 8th International ESA Conference on Guidance, Navigation & Control Systems*. Karlovy Vary, Czech Republic, 2011.

[209] S. Paschall and T. Brady. "Demonstration of a Safe & Precise Planetary Landing System On-board a Terrestrial Rocket". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2012.

[210] S. Paschall and T. Brady. "Rocket validation of the ALHAT autonomous GNC flight system". In: *Proceedings of the IEEE Aerospace Conference*. 2014.

[211] T. J. Steiner, T. M. Brady, and J. A. Hoffman. "Graph-based Terrain Relative Navigation with Optimal Landmark Selection". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2015.

[212] A. Katake et al. "LandingNav: a precision autonomous landing sensor for robotic platforms on planetary bodies". In: *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics. 2010, pp. 75390D–75390D.

[213] F. Amzajerdian et al. "Doppler Lidar Sensor for Precision Landing on the Moon and Mars". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2012.

[214] D. Adams, T. B. Criss, and U. J. Shankar. "Passive Optical Terrain Relative Navigation Using APLNav". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2008.

[215] C. Cocaud and T. Kubota. "Autonomous Navigation Near Asteroids Based on Visual SLAM". In: *Proceedings of the 23rd International Symposium on Space Flight Dynamics*. Pasadena, California, 2012.

[216] J. Alexander et al. "A Terrain Relative Navigation Sensor Enabled by Multi-Core Processing". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2012.

[217] B. Van Pham et al. "Landmark Constellation Matching for Planetary Lander Absolute Localization." In: *Proceedings of the International Conference on Computer Vision Theory and Application*. 2010, pp. 267–74.

[218] B. Van Pham et al. "Fusion of absolute vision-based localization and visual odometry for spacecraft pinpoint landing". In: *Proceedings of the 9th International Planetary Probe Workshop*. Barcelona, Spain, 2010.

[219] J. Cuseo et al. "Machine Vision Techniques for Planetary Terminal Descent Hazard Avoidance and Landmark Tracking". In: *Proceedings of the American Control Conference*. 1991, pp. 1406–7.

[220] C. C. Liebe. "Tracking of planetary terrains". In: *IEEE Aerospace and Electronic Systems Magazine* 9.2 (1994), pp. 9–18.

[221] A. I. Mourikis et al. "Vision-Aided Inertial Navigation for Spacecraft Entry, Descent, and Landing". In: *IEEE Transactions on Robotics* 25.2 (2009), pp. 264–80.

[222] S Thurman et al. "Space Flight Test of Vision-Guided Planetary Landing System". In: *Proceedings of the AIAA Infotech@ Aerospace Conference*. 2007.

[223] N. Trawny et al. "Vision-aided inertial navigation for pin-point landing using observations of mapped landmarks". In: *Journal of Field Robotics* 24.5 (2007), pp. 357–78.

[224] A. I. Mourikis and S. I. Roumeliotis. "A multi-state constraint Kalman filter for vision-aided inertial navigation". In: *IEEE international Conference on Robotics and Automation*. 2007, pp. 3565–72.

[225] A. Mourikis et al. "Vision-Aided Inertial Navigation for Precise Planetary Landing: Analysis and Experiments". In: *Proceedings of Robotics: Science and Systems*. Atlanta, GA, USA, 2007.

[226] J. Delaune et al. "Optical Terrain Navigation for Pinpoint Landing: Image Scale and Position-Guided Landmark Matching". In: *Proceedings of the 35th Annual AAS Guidance and Control Conference*. Brekenridge, Colorado, 2012.

[227] L. Singh and S. Lim. "On Lunar On-Orbit Vision-Based Navigation: Terrain Mapping, Feature Tracking Driven EKF". In: *Proceedings of the AIAA Guidance, Navigation and Control Conference*. Honolulu, Hawaii, 2008.

[228] A. E. Johnson and J. F. Montgomery. "Overview of Terrain Relative Navigation approaches for precise lunar landing". In: *Proceedings of the IEEE Aerospace Conference*. Big Sky, Montana, 2008.

[229] F. Dellaert and M. Kaess. "Square Root SAM". In: *Robotics: Science and Systems*. 2005, pp. 177–84.

[230] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519.

[231] F. Dellaert. *Factor graphs and GTSAM: A hands-on introduction*. Tech. rep. 2012.

[232] M. J. Powell. "A new algorithm for unconstrained optimization". In: *Nonlinear programming* (1970), pp. 31–65.

[233] M. Powell. "On the global convergence of trust region algorithms for unconstrained minimization". In: *Mathematical Programming* 29.3 (1984), pp. 297–303.

[234] M. Graham and J. How. "Robust Simultaneous Localization and Mapping via Information Matrix Estimation". In: *ION/IEEE Position, Location and Navigation Symposium*. 2014, pp. 937–44.

[235] E. Olson and P. Agarwal. "Inference on networks of mixtures for robust robot mapping". In: *The International Journal of Robotics Research* 32.7 (2013), pp. 826–840.

[236] N Sunderhauf and P. Protzel. "Towards a robust back-end for pose graph SLAM". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2012, pp. 1254–61.

[237] Y. Latif, C. Cadena, and J. Neira. "Robust loop closing over time for pose graph SLAM". In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1611–26.

[238] C. Ake, J. Scotkin, and D. Masten. "Exploring the benefits of commercial robotic lander testbeds". In: *Proceedings of the IEEE Aerospace Conference.* 2012.

[239] T. J. Steiner. "A Unified Vision and Inertial Navigation System for Planetary Hoppers". MA thesis. Massachusetts Institute of Technology, 2012, p. 146.

[240] R. Lerner, E. Rivlin, and I. Shimshoni. "Landmark selection for task-oriented navigation". In: *IEEE Transactions on Robotics* 23.3 (2007), pp. 494–505.

[241] P. Sala et al. "Landmark selection for vision-based navigation". In: *Robotics, IEEE Transactions on Robotics* 22.2 (2006), pp. 334–49.