COHESIVE AUTONOMOUS NAVIGATION SYSTEM: IMAGE PROCESSING AND DATA MANAGEMENT

A Thesis

by

DEREK J KUETHER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2016

Major Subject: Aerospace Engineering

ABSTRACT

The ability of a robotic system to fully and autonomously interact with its environment is key to the future of applications such as commercial package delivery services, elderly robotic assistants, agricultural monitoring systems, natural disaster search and rescue robots, civil construction monitoring systems, robotic satellite servicing, and many more. An architecture that is conducive to Simultaneous Localization And Mapping (SLAM), path planning, and mission planning is a critical element for a system to be robust enough to handle such applications with true autonomy.

In this work, an architecture that lends itself to cohesive operation of all the aforementioned goals is presented. The key components of this architecture are the data management, image processing, SLAM, and path planning. The architecture works through the implementation of a common core database to represent the environment. The database management tools use k-vector search techniques. Image processing techniques are evaluated in a trade study where a graph-based approach is selected. An outline of the SLAM approach and a description of the path planning algorithm employed are briefly discussed. The Cohesive Autonomous Navigation System (CANS) is successfully implemented at slower than real-time speeds and future work is outlined to achieve a real-time system.

ACKNOWLEDGEMENTS

I would like to thank first and foremost my colleagues Benjamin Morrell and Mauricio Coen. They contributed just as much as I did into the development of the work discussed in this thesis. I would also like to thank my advisor Dr. Gregory Chamitoff for his invaluable guidance, critism, and encouragement in working through many problems throughout this work. The contributions of my committee (Dr Daniele Mortari, Dr Michael Bishop, and Dr Gregory Chamitoff) provided much of the fundamental methods and tools that were expaned/tested here. A large part of the work presented here would not have been possible if it weren't for the Land, Air, and Space Robotics (LASR) laboratory. I would also like to thank the many others that discussed aspects of the problem and assisted in one way or another: PhD candidate Joseph S. Lee, Dr. Dezhen Song, Dr. Steve Liu, Chip Hill, Dr. Peter Gibbens, PhD candidate Austin Probe, and many others.

Of course, an acknowledgments section would not be complete without acknowledging my loved ones - my mother, Dawn, father, Jim, step-father, Cal, sister, Kendra, brother, Austin, and my supportive girlfriend, Chrisy. A particular dedication is directed to my brother Austin who will forever be remembered as the gentle giant he was.

I hope you enjoy this thesis as much as it excites me.

NOMENCLATURE

Agent	Operational sequence.
CANS	Cohesive Autonomous Navigation System.
Channel	An image holding a metric value.
CIE	International Commission on Illumination.
Feature	A modeled entity that exists in physical space.
GIS	Geographic Information Science.
Label Map	An image with labels, integers, where each label indicates which group each pixel belongs to.
LOI	Label of Interest: Label of the label map indicating which index of the label map indicates a region to be modeled as an ellipsoid.
Luv	Color space develeloped by CIE.
Metrics	Random variable representing a descriptor of the pixel of an image.
Object	See Feature.
OLT	OverLaid Thresholding: Image segmenation technique.
RGB-D	Set of channels with red intensity, green intensity, blue intensity, and depth information of a field of view.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION*

I.A. Motivation

The ability for a robotic system to fully and autonomously interact with its environment is key to the future of applications such as commercial package delivery services, elderly robotic assistants, agricultural monitoring systems, natural disaster search and rescue robots, civil construction monitoring systems, robotic satellite servicing, and many more. An architecture that is conducive to Simultaneous Localization And Mapping (SLAM), path planning, and mission planning is a critical element of a system to be robust enough to handle such applications with true autonomy. Such a system involves an image processing methodology to extract relevant and concise information from the environment. This methodology needs to be consistent and robust to varying lighting conditions, types of objects in each scene, and to represent sufficiently unique distinctions in the scenes. Additionally, the data management scheme must be optimized for efficient utilization by SLAM processes, path planning processes, and mission planning processes. Image processing and data management are two key elements of a Cohesive Autonomous Navigation System (CANS) that will be the focus of this study.

*Portions of this chapter are reprinted with permission from "Cohesive Autonomous Navigation System" by Kuether, Derek J., et al., 2016. AIAA Guidance, Navigation, and Control Conference, AIAA SciTech, Copyright 2016 by American Institute of Aeronautics and Astronautics, Inc.

I.B. Outline

In this thesis, an architecture developed with my colleagues [1] will be discussed. This architecture lends itself to cohesive operation of all the aforementioned goals through the implementation of a common core database to represent the environment. The key contributions to this architecture that are presented here are in the areas of Image Processing and Data Management.

Section (II) will give brief overview of CANS. Section (III) will discuss the image processing module in detail. Section (IV) will discuss the data management module in detail. Section (V) will discuss the entire system's performance. Section (VI.A) presents conclusions. Section (VI.B) will highlight areas of future work and reference information is presented in several appendices in Section (VI.B.3).

All the results presented here were run on a Yoga 2 Pro laptop. Additional specifications are in Table (I.1).

I.C. Background

Key to fully autonomous robotic systems is the ability to navigate through an environment without any prior knowledge. This can be broken down into three operations: creating a map of your environment, finding out where you are in the environment, and planning a time-dependent path[1] through your environment. The first two operations are typically done via a methodology called Simultaneous Localization And Mapping (SLAM) [2, 3]. Path planning and SLAM are generally

[1]From here on, trajectory and time-dependent path will be used interchangeably.

Table I.1. CANS simulation environment specifications

Parameter	Value
Make	Lenovo
Model	Yoga 2 Pro
Operating System	Ubuntu 15.10
Operating System Type	64-bit
Memory	7.7 GiB
Processor	Intel Core i7-4500U CPU
Clock Speed	1.8 GHz x 4
Graphics	Intel Haswell Mobile
Disk	243.5 GB

considered as two separate problems. SLAM looks at an environment and localizes itself within it while developing a model for the environment. This process is circular: localization in the map depends on the current model, and generation of the model depends on the current location. This process is most commonly tackled with some variation of a Kalman filter, including the Extended Kalman Filter [2], the Unscented Kalman Filter [4], and the Ensemble Kalman filter [5]. Major distinguishing factors in different SLAM techniques are: the choice of Kalman filter (or an alternative estimation method [6, 7]), the nature of the features used as landmarks and how the features are matched between two observations. Path planning, on the other hand, takes a desired objective and plots a trajectory to obtain the objective through the known environment without violating constraints on the path, such as the physical obstacles. There are many different approaches to path planning, such as A* algorithms [8], Mixed Integer Programming [9], potential field minimization [10, 11], parameter optimization [12] and a suite of sampling based planners like Rapidly Exploring Random Trees [13] (many available in the Open Motion Planning Library [14]), each with their own strengths and weaknesses.

For a system to be truly autonomous, it needs to be capable of both SLAM and path planning. So intuitively, it makes sense to develop an architecture which is conducive to the efficient joint operation of the two processes. However, the analyses of many SLAM techniques are demonstrated with the performance of just the standalone process [15, 16]. Similarly for path planning, approaches generally are demonstrated with a known environment. There are key points that make combining the two processes difficult. SLAM typically uses point locations as features to localize the system and to match with previously observed features, a process termed data association. Path planning necessitates a full 3-dimensional physical representation of all features to define physical constraints. The key approach taken in this system is to combine these elements by defining an observed feature with both a point location as a centroid and a 3D physical description that can be used for data association *and* as an obstacle for the path planner. Here ellipsoids and rectangular prisms (patches) are used to represent these 3D features. They are convex shapes and relatively easy to describe.

To have a 3D description of the environment, it is beneficial to have sensors that can provide 3D information, such as stereo cameras, monocular cameras with egomotion, laser scanners or structured light sensors, such as the Kinect RGB-D (color image and depth) sensor [17], which also provides RGB (color) information. The active approaches (laser scanners and structured light sensors) can provide more precise information for characterizing the environment, and hence are used in the work presented in this thesis. Specifically, an ASUS Xtion Pro will be used in simulation.

The criteria that identify points of interest to SLAM are often very simple metrics that can be correlated to a point. Metrics such as color and texture do not make physical sense for a point and are not unique when describing features by points. While these point objects are often easy to identify, they are susceptible to being associated with the incorrect feature in the database, which corrupts the SLAM solution. By using color and texture to provide more complex representations of features, the association between features becomes more certain. Features can be described with location, size, orientation and color as well as structural information that can be computed from a wide range of surface analysis algorithms.

Given some set of images, say color images (RGB) and depth image, there are numerous methods to identify different distinct regions (or perform image segmentation). Generally speaking, they can be distinguished into two categories. These are edge/threshold based detection algorithms and homogeneity algorithms. Edge-based algorithms include Watershed [18], Oksu's method [19], and overlaid thresholding. Homogeneity algorithms include K-Means [20], graph-based [21], and mean-shift [22]. It is important to distinguish these algorithms from pattern recognition. Segmentation algorithms themselves do not identify objects in scenes (as is the case with pattern recognition); rather, they distinguish what data points belong to a different object. The identity of the object is undefined.

A unique aspect of CANS is that GIS (Geographic Information Science) surface feature analysis tools have been applied to the problem of autonomous robot navigation. These GIS analytical techniques are extensively used to describe landscapes with numerous metrics such as convergence, divergence, roughness, curvature, and vegetation indices. [23, 24] In many cases, these descriptors are used to identify areas of interest, and uniquely describe features. In addition to providing information to define distinct features, the GIS information also lends itself to being used in mission planning, by classifying target areas of interest. Recording these target areas will provide CANS with the necessary information to dynamically reassess the mission and potentially redirect its mission objectives. Using complex, multi-dimensional features helps make each feature unique, which enhances the certainty of associations and thus certainty in state estimates. Extracting and matching these complex features can become computationally expensive though, and hence requires a trade-off between highly descriptive features and features that are computationally efficient. The proposed architecture is designed to aid in this trade-off by using a database structure and rapid search algorithms to facilitate efficient matching of features described by multiple parameters. The search techniques utilize the novel k-vector method, a technique that rapidly performs a range search that is independent of database size [25].

I.D. Contributions of Work

Autonomous navigation systems are not a new field of study, so what is the contribution of this work? CANS hosts several key aspects that make it a novel system. First, CANS uses 3D features for SLAM (and the rest of the system). Second, CANS uses a set of GIS information for operational means. Third, k-vector search techniques are employed for database operations. In the journey to this end, lessons are learned and the experiments conducted provide some interesting information.

CHAPTER II

CANS OVERVIEW

CANS is designed to maintain a set of software agents to interact between large operations. These large operations, termed modules, consist of SLAM, image processing (including feature extraction and matching), and path planning as can be seen in Figure II.1.

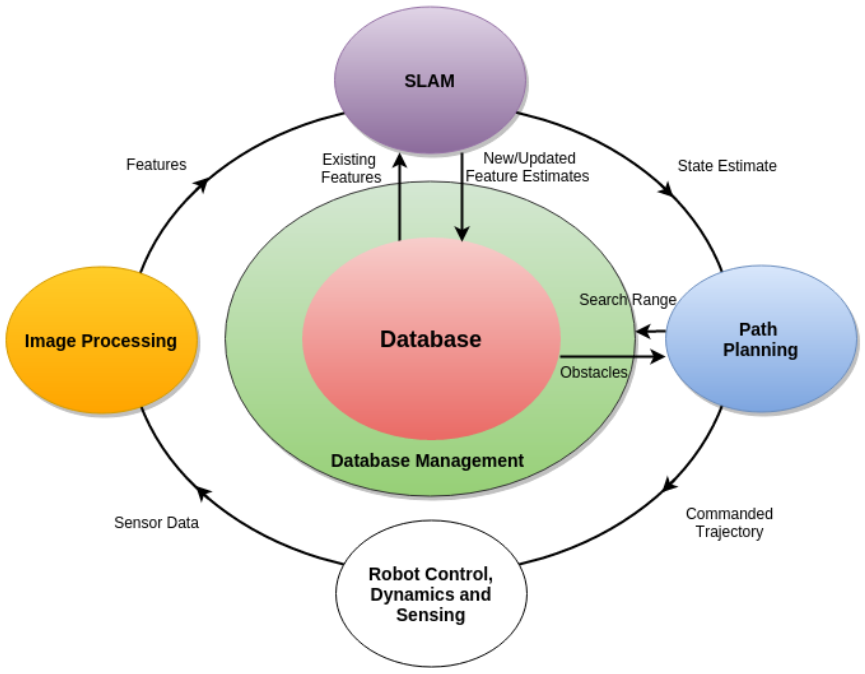


Figure II.1. System overview

The image processing module takes the sensor information to extract and describe features in the environment. These extracted objects in the environment are then passed to the SLAM module. The SLAM module takes the observed features and passes search parameters based on each object to the database management module. Suppose an image is captured viewing a water bottle lying on a table. The image processing module will take this information and represent the water bottle with 3D feature. The parameters of this 3D feature are passed to the SLAM module which searches the database for a feature just like it.

The database management module passes back found objects. SLAM provides the database management module updated features and new features to update and add to the database. Continuing with the example, the database will pass back the matched feature that represents the water bottle if it exists. SLAM will update the feature with its new estimate of the parameters and any other new features observed.

SLAM passes a system state estimate to the path planning module. The path planning module will pass search parameters to the database management module to extract physical objects in the region where the trajectory is being planned. The database management module returns the set of objects in the region of interest. The path planning algorithm then plans the path and sends the waypoint trajectory to the robot control, dynamics, and sensing module for the robot to follow. Finishing with the same example, the SLAM module has a new estimate of where the system is based on how the water bottle moved in its field of view. The path planning module will request the database for all features in the space it is planning a path. Say the database returns the water bottle because it is in this space. The path planning algorithm will generate a path around the water bottle and provide this commanded path to the robotic control, dynamics, and sensing module.

CHAPTER III

IMAGE PROCESSING

III.A. Overview

The image processing module extracts distinct features from sensor images and describes them with physical metrics (centroid, size, orientation) and additional metrics (color, texture, slope, other GIS indices). To do this, the image undergoes a series of operations. Figure III.1 illustrates the process of image processing. Each step of the flow diagram will be discussed below followed by an illustration of the transition between each intermediary step in a full image processing cycle.

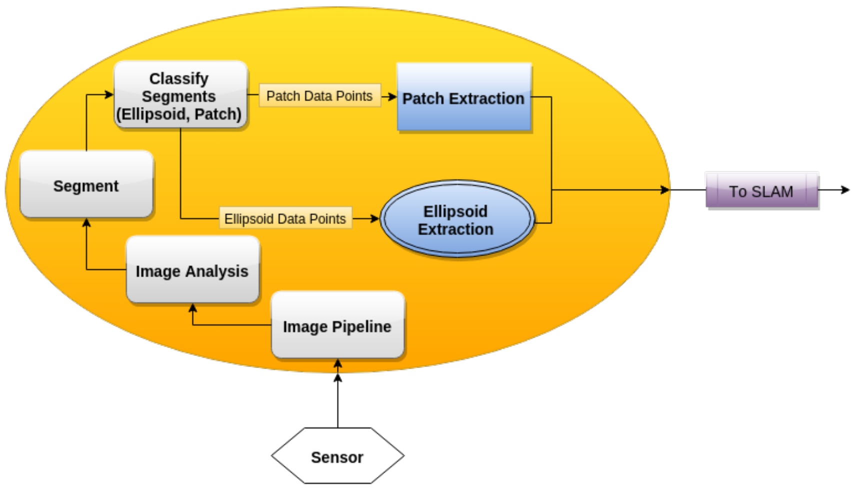


Figure III.1. Image processing overview

III.A.1. Image Pipeline

To prepare the sensor data for the operations done in image processing, the raw information gathered must be aligned. All the metrics gathered for each image must align with the appropriate pixel. For example, the infrared camera of an ASUS Xtion sensor does not perfectly align with the RGB camera. The image alignment operation corrects the metric maps to align each pixel with the same point in physical space.

III.A.2. Image Analysis

As one of the most beneficial characteristics of CANS, we can describe a feature with many metrics. To do so we run the raw RGB-D through several filters to produce channels of data. Here, a channel refers to a single image (pixel map, with metric values assigned for each pixel) consisting of a single metric (e.g., X coordinate, red intensity R, Z coordinate derivative, etc.). We will briefly discuss these filters.

The raw spatial data obtained by the ASUS Xtion sensor is a disparity map. The disparity map is used to calculate the depth map in the RGB-D information. The OpenNI library [26] is use to calculate the depth to each pixel, and then the corresponding X, Y, and Z coordinates to give a 3D point cloud.

Although the RGB color space is very intuitive, it may not be ideal for SLAM. For example, if the lighting conditions between image acquisitions are inconsistent, the intensity of red, blue, and green will be different for the same object. This may result in an false negative data association. A more appropriate color space may be the CIE-Luv [27, 28], where L stands for lightness. This color space has a dimension (L) that is linear in lightness. Thus, a change in the shading of the object could be directly characterized by L. This color space standard is also one of the two suggested standards by CIE (International Commission on Illumination).

The field of Geographic Information Science (GIS) uses a suite of algorithms to determine characteristics of an environment such as slope, roughness, curvatures, and vegetation indices [23, 24]. These metrics form what the GIS community call layers of maps, where each map contains a different metric for the pixels (here we are instead referring to layers as channels). Using these metrics, one can describe mountains, lakes, corn fields, forests, etc. Simply overlapping these layers can highlight regions otherwise indistinguishable by the human eye. This becomes especially true when many of the metrics are determined from hyperspectral data. The ease with which GIS algorithms can identify key areas in an image, and later in a 3-dimensional world, make them excellent tools for feature identification. Thus, we utilize channels of metrics to determine the features of interest, data association, and a potential tool for science driven missions. In this work, the following GIS indices will be discussed:

- Slope Azimuth Divergence Index
- Slope Azimuth Convergence Index
- Slope Azimuth Homogeneity Index
- Surface Roughness Factor
- Tangential Curvature
- Planimetric Curvature
- Profile Curvature
- Unsphericity Curvature
- Difference Curvature
- Minimal Curvature
- Maximal Curvature
- Horizontal Excess Curvature
- Vertical Excess Curvature
- Mean Curvature

These metrics represent what their names might imply. The way these metrics are calculated is protected by a non-disclosure agreement and thus are not presented here.[1]

[1]If a determined reader is interested in the details of these metrics, they should refer to Dr. Michael Bishop.

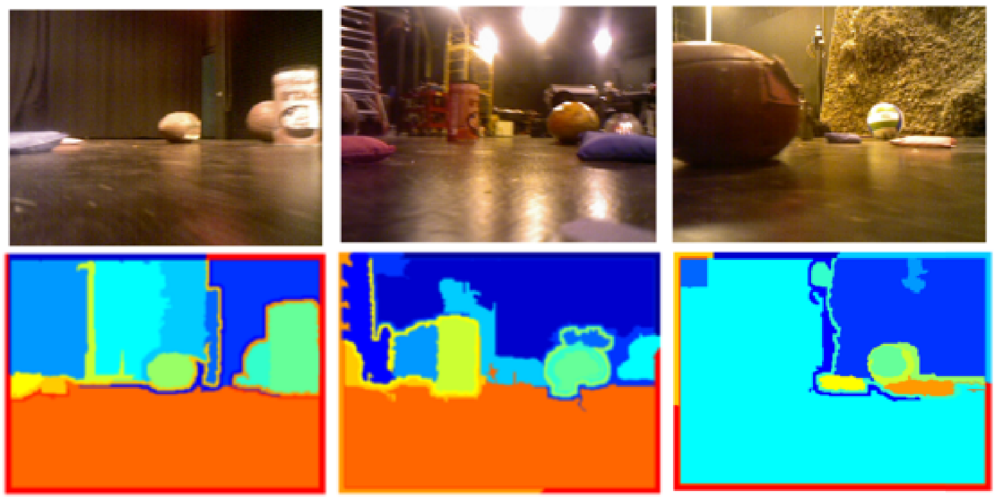


Figure III.2. Example of segmentation. Top: original image. Bottom: corresponding segmentation.

III.A.3. Image Segmentation

Though we are modeling the environment with primitive shapes (ellipsoids and simple prisms), the criteria to form the ellipsoids is based on more than just capturing all the physical, 3D characteristics observed. Instead, we are interested in the most unique features in the environment.

Figure III.2 shows three examples of what the segmentation may look like. The top row shows the original color image for an image acquisition. The bottom row shows the segmentation for the corresponding color image directly above it. The distinct regions are indicated by a different color. This figure is used to illustrate how the regions can be visualized. In reality, each different color represents a integer. The pixels in each channel have a corresponding label pixel in this label map. All the pixels with the same integer belong together and so do the corresponding pixels in each channel.

The reader should note that the balls in the left-most image are clearly segmented into distinct regions. The reader should also notice that the curtain in the back is broken into seperate regions. Here lies a major challenge in this step. There are many different ideas about what should be identified as distinct regions and then how to conduct the segmentation (as discussed in Section (I.C)). One method may work well in a specific environment, but finding a method that works for the more general solution is difficult. On the right-most image, the reader will clearly see a football (perhaps not so obvious it is a football). However, the segmentation below shows no indication of it. This is due to the football being too close to the sensor, thus providing null values for spatial information[2]. How to deal with imperfect or very restricted information provides another challenge. In a more fully developed system, CANS may wish to discern valuable differentiations between areas despite invalid spatial information. Thus, this leads to another challenge.

[2]The sensor has a limited range. Outside the bounds of this range generates null or invalid depth information.

III.A.4. Feature Classification

To be completely autonomous, the system must be able to identify which regions are of interest for SLAM, path planning and mission planning. These regions are represented by their labels in the label map and are aptly called Labels of Interest (LOIs). In a system that uses RGB-D information, all regions are of interest to path planning because they represent physical constraints that need to be avoided. In order to determine what regions are of interest to SLAM, we run all the regions through a series of predicates.

$$P_1 = (Min\ Pixel\ Count < size(v_i)) \tag{3.1}$$

$$P_2 = (Max\ Pixel\ Count > size(v_i)) \tag{3.2}$$

$$P_3 = (Max\ Invalid\ Pixel\ Count > Invalid(v_i)) \tag{3.3}$$

$$P_4 = (Max\ Distance > norm(v_i(X), v_i(Y), v_i(Z))) \tag{3.4}$$

$$P_5 = (Min\ Distance < norm(v_i(X), v_i(Y), v_i(Z))) \tag{3.5}$$

where v_i is the set of pixels for node i of the reduced graph [3]. The predicate in Eq. (3.1) provides a lower bound on the number of data points that must exist in a region to be considered as an ellipsoid candidate. The predicate in Eq. (3.2) provides an upper bound on the number of data points that must exist in a region to be considered as an ellipsoid candidate. The data that we receive from the ASUS Xtion is not high quality, hence there is likely to be invalid data. Invalid data is also

[3]Nodes of the graph represent regions of the graph that represents the iamge. A region may contain one pixel or many more. The reduced graph is the graph of nodes less than the number of total pixels, representing the image of segments.

introduced when the objects in line-of-sight are out of range for the IR camera. A feature must be modeled with spatial validity; to the contrary would create inaccurate estimates of physical metrics (such as centroid) which introduce errors in the SLAM process. Thus, the predicate in Eq. (3.3) provides an upper bound on the number of invalid data points allowable in an ellipsoid candidate.

If a feature passes all these predicates, we model the set of data points with an ellipsoid. For features that do not pass all the predicates, they are labeled patches. These patches (each a set of data points) are non-unique. The primary purpose of the patches is to represent a physical obstruction for the path planning algorithm. Thus, the patches are modeled as simple prisms with no regard for their boundaries introduced via the segmentation process. The patch modeling is current work in progress.

III.A.5. Feature Extraction[4]

The feature extraction step takes the regions of interest and extracts a physical description of the feature from the corresponding 3D point cloud. The goal of the feature extraction step is to quickly generate a representative physical description of the feature that will be consistent through observations from different locations, and hence be useful for data association when making multiple observations over time. The defining descriptors are the location, the orientation and the size. We use ellipsoids to represent features by these physical aspects, described by the centroid, three orthogonal magnitudes and an orientation described by quaternions. One possible

[4]For completeness, here I will discuss some work conducted by my colleague Benjamin Morrell.

approach to compute this description is to compute the covariance matrix associated with the 3D points, giving the orientation of the ellipsoid, and use an $n \times \sigma$ bound to set the size of the ellipsoid (where n is a tuning parameter, and σ the standard deviation). This approach is not taken, as it requires a Gaussian assumption on the distribution of 3D points. While this might be well suited for observing ellipsoids, it is an undesirable assumption for observations of the surface of general shapes. Instead we use an averaging approach to capture the 3D structure of the observed point cloud. The first step is computing the centroid, by taking the mean of the 3D point cloud associated with the region of interest. Then a percentage (25% here) of points furthest from the mean are excluded in the ellipsoid extraction. This is done to ensure that outlier points do not skew the ellispoids. These outlier points may belong to background features that were incorrectly assigned to the feature being extracted. A series of simple computations are then employed to rapidly determine the size and orientation of the ellipsoid. First, the region around the centroid is split into 16, partially overlapping regions. The first 8 regions are given by the octant attained from the division of 3D space by the coordinate axes. The second 8 are the division of 3D space by the coordinate axes transformed by two 45 deg rotations around the x and y axes respectively. The intention is to have a simple division of 3D space, and avoid any common dividing lines between regions, where the averaging approach described below would misrepresent the physical object. For each region, a sum of the points in that region, weighted by the distances from the centroid is computed:

$$M_i = \sum_{\forall p \epsilon R_i} d_p^2 \tag{3.6}$$

where M_i is the distance square sum for each segment R_i, p denotes each point and d_p is the distance of each point from the centroid. The vector from the centroid to the mean of the points in the region with the largest M_i gives the direction of the largest ellipsoid axis, which we call the *primary axis*. The process is repeated for two overlapping quadrant sets around the primary axis, giving 8 regions. In this case, the d_p in equation 3.6 represents the perpendicular distance from the primary axis. The vector from the centroid to the mean of the points in the region with the largest M_i is again used, but gives a vector whose component perpendicular to the primary axis gives the direction of the *secondary axis*. The final, *tertiary axis*, direction is given by a cross product of the primary and secondary axes to complete a right handed set. The orientation of this right handed set with respect to the camera frame gives the orientation description of the ellipsoid, and is described with quaternions.

The size of the ellipsoid is represented by distances along each of the three axes (as the semi-major axes of the ellipsoid). Each axis magnitude is the magnitude of the standard deviation in the corresponding R_i, scaled by a factor α. By tuning α, the ellipsoid size can be changed to balance the inclusion of some outlying points, with being too large of an ellipsoid.

Figure III.3 shows three simple examples of ellipsoid generation, where point clouds observations (blue dots) are modeled from a view of ellipsoids in physical space (red), and are fit with ellipsoid representations (black). Only one face of the red ellipsoid is visible, hence points on only one face can be observed. A closer match

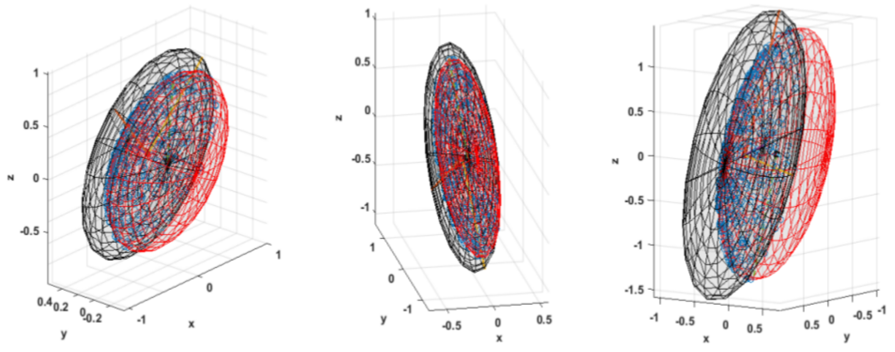


Figure III.3. Ellipsoid generation from observations of ellipsoids

to the red ellipsoid cannot be achieved without making assumptions on the nature of the true object: an undesirable step for generalization to a range of objects in the environment. Nonetheless, the generated ellipsoid gives a good representation of the shape, size and location. With multiple observations merged together, the observed ellipsoids (black) will evolve to more closely match the true ellipsoids (red).

The results from a true point cloud are shown in Figure III.4. The results show an ellipsoid extracted from the region of interest of a filing cabinet in the foreground (represented by the purple points in Figure III.4, left). The right image in Figure III.4 shows just the points and the ellipsoid. The ellipsoid shows a good representation of the orientation and size of the feature, but balanced with encompassing a majority of the observed points (93.43% of the points, with $\alpha = 4$). Parameter α could be modified to encompass a greater percentage of the points.

To ensure that the ellipsoids created are valid, the extracted ellipsoids are passed through a second series of predicates pertaining to the ellipsoid characteristics. These are the predicates listed in Eqs. (3.7) to (3.10). As can be seen in Figure III.2, there are regions that span a large portion of the image and lie on the perimeter of a series

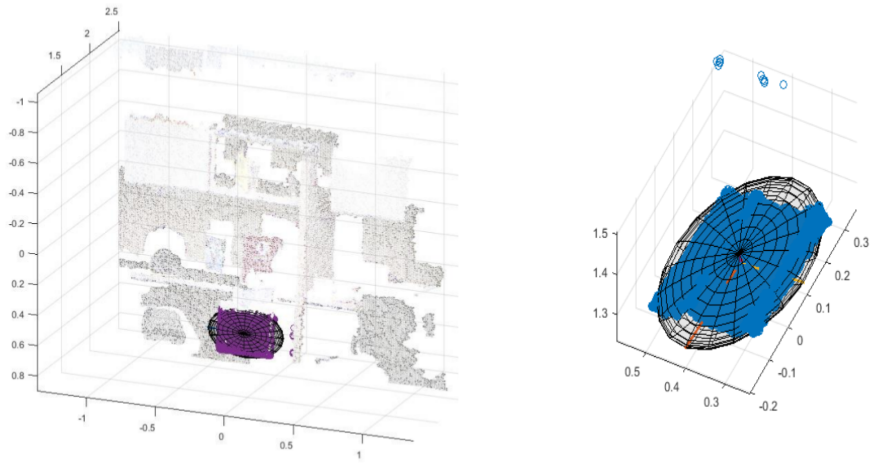


Figure III.4. Ellipsoid extraction from cabinet feature in point cloud image. Left: with full point cloud. Right: only the region of interest.

of physical objects. Additionally, a segmentation may merge a large set of outlier pixels in the background to a foreground object. The result would skew the resulting ellipsoid. As a means to remove these potential issues, the predicate in Eq. (3.7) is used. Note that the seconday axis is used here. We do not want to exclude ellipsoids that are relatively planar. This will often be the case when observing a side of flat object since we have no information about how deep the object actually is.

$$P_6 = \left(Max\ Aspect\ Ratio > \frac{Major\ Axis\ Magnitude}{Secondary\ Axis\ Magnitude} \right) \tag{3.7}$$

$$P_7 = (Max\ Axis\ Magnitude > Major\ Axis\ Magnitude) \tag{3.8}$$

$$P_8 = (Min\ Axis\ Magnitude < Minor\ Axis\ Magnitude) \tag{3.9}$$

$$P_9 = (Min\ Largest\ Axis\ Magnitude < Major\ Axis\ Magnitude) \tag{3.10}$$

Predicates in Eqs. (3.8) to (3.10) ensure that no abnormal ellipsoids are recorded. Potential abnormal ellipsoids include partial segmentations of larger objects about a

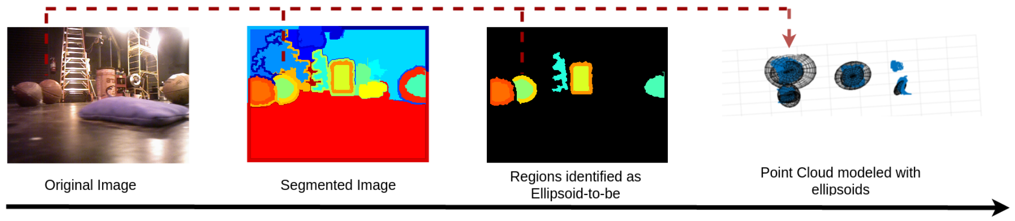

Original Image
Segmented Image
Regions identified as Ellipsoid-to-be
Point Cloud modeled with ellipsoids

Figure III.5. Sample image processing cycle

shadow or about a reflective edge.

III.A.6. Image Processing Cycle

Figure III.5 illustrates the entire image processing cycle from the raw data (the color image on the left in addition to the depth map) to the set of ellipsoid models (right most image) of regions extracted as ellipsoid candidates. Recalling the previous sections, the first image on the left is the color image collected from the sensor (a disparity map is also collected). The second image is the result of the image segmentation. Note that this step follows the image analysis. The third image highlights the regions that passed the predicates - both before and after the ellipsoid extraction. The fourth image displays the point cloud and encapsulating ellipsoids formed by the feature extraction. The descriptors of these ellipsoids and average of their metrics for the points in the objects are passed as outputs of the image processing module.

Note that there is more work to be done on the image processing to achieve the level of performance desired. There are artifacts created in the image segmentation that continue on to be selected as ellipsoid candidates. A more thorough discussion of these challenges and future work will be discussed in section VI.B. For a more interactive demonstration of the image processing, please refer to the accompanying video[5] of a full test set with the associated segmentations and extracted regions. Thus far, an overview of the image processing module has been discussed. In Section (III.B), an overview of different options for image segmentation will be discussed. In Section (III.C), a trade study of these methods will be presented. In Section (III.D), one method will be selected for futher investigation to suit the task at hand.

III.B. Segmentation Methods Investigated

The methods presented here, with the exception of OverLaid Thresholding (OLT), are popular methods used in the field. This section will discuss each one in some detail. The level of detail is presented to allow understanding of following analysis. Further details are available in the referenced works. Following an overview of all the methods, Section (III.C) will investigate different performance parameters.

III.B.1. Watershed

The watershed algorithm, as aptly named, is analogous to the basins and ridges of a watershed. The algorithm is explained in much more detail in Meyer (1994) [18]. In basic principle, the algorithm follows the ridges of the image (i.e., the high value, 0 gradient pixels), creating branching edges accross the image. Figure (III.6) shows the color image of a sample measurement. This measurement is accompanied with a depth map. Figure (III.7) shows a sample segmentation of this image.

[5]Image Processing Animation. URL: `https://www.youtube.com/watch?v=jA-yeVXIbG4`

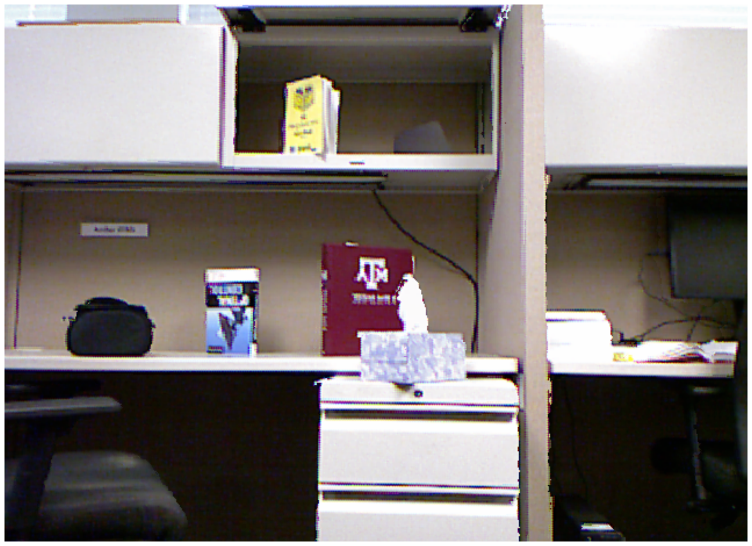


Figure III.6. Sample image processing image

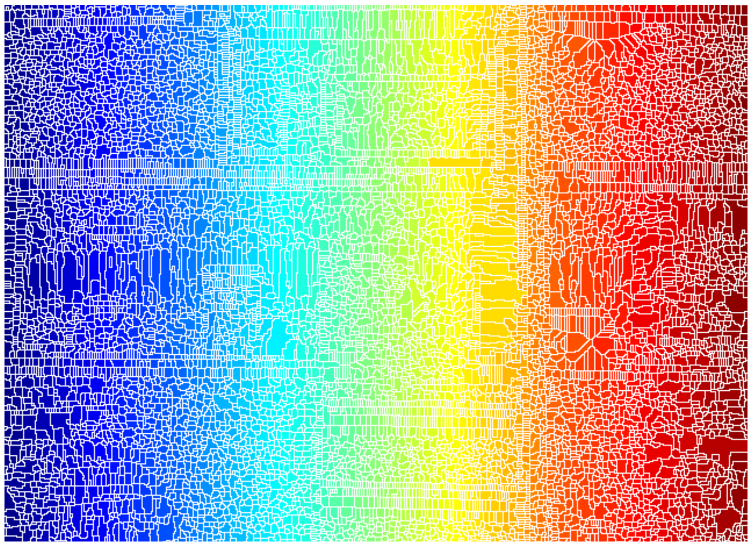


Figure III.7. Sample image segmentation - watershed algorithm

Holding off on any premature conclusions about the algorithm, we can see that the image is segmented into numerous different regions, where each region is represented by a different color. Due to the high number of segmentations, many of the colors look alike. It is perhaps easier to distinguish between different regions by the white lines seperating each region. Also note that the segments do not form a boundary around any intuitive features in the environment. The watershed algorithm is performed on a single grayscale image. A reduced number segmentation can be produced if the image is smoothed.

III.B.2. Overlaid Thresholding

Another method investigated in this study is an algorithm called Overlaid Thresholding (OLT) - created in this work. The algorithm is composed of two phases. A primary segmentation followed by a region merging algorithm. The primary segmentation is used to break up the image into regions of similarity. To define areas of similarity, we threshold (θ) all the curvature of all the metrics' values and overlay them on each other. The threshold is defined in Eq. (3.11).

$$\theta = \bar{M} + \alpha\sigma_M \tag{3.11}$$

where M:=An image of metrics (for example, red intesnity image), α:=A tunable constant factor, $\bar{M}$:= Mean of image metric, M, and σ_M :=Standard deviation of image of metric, M. After thresholding the curvature metrics, we reduce the resulting binary image into a skeleton[6] (M_S) by the Zhang-Suen thinning algorithm [29]. Each

[6]Given a binary image where 0's represent edges, the original image has edges multiple pixels wide. The skeleton of this image is one such that these wide edges are reduced to single pixel width.

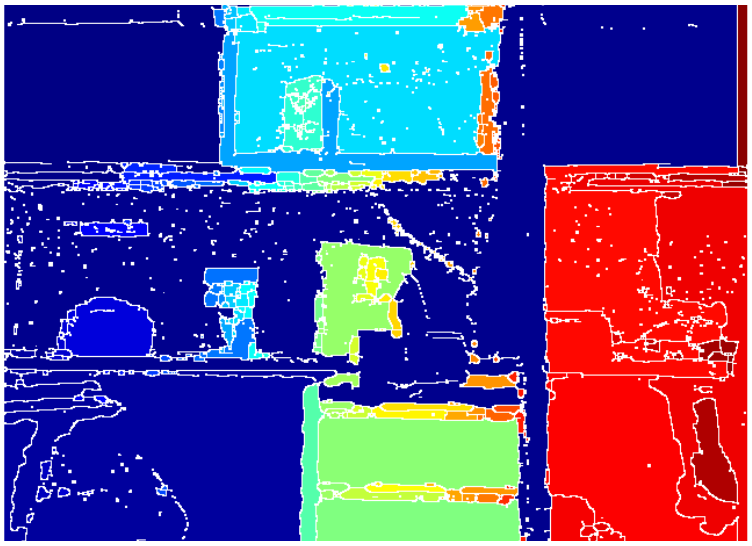


Figure III.8. Sample segmentation: OLT (without N-CUT)

metric skeleton is overlaid on each other so that we create a binary image, B, that follows the rule in Eq. (3.12).

$$B(i,j) = \prod_{k=1}^{\lambda} M_s(i,j,k) \tag{3.12}$$

where λ:=number of metrics. The binary image B is an initial segmentation of the image. Recall the sample image in Figure (III.6).

Figure (III.8) shows the segmentation of Figure (III.6) from the skeleton image produced by Eq. (3.12). As can be noticed by inspecting Figure (III.8), several of the regions break apart 'intuitive' segmentations. Notice, for example, the book on the left in the image. The segmentation breaks apart the image into several parts.

To address this, the primary segmentation is coupled with a region merging algorithm called N-Cut [30]. Region merging is used to take the primary segmentation and merge regions that are alike enough to be considered the same region. To do this, we create a graph model (G) of the labeled image. $G = (V, E)$, where V is the set of nodes or regions of the image and E is the set of edges of the graph that can be represented as an affinity matrix[7] such that $e(i, j)\epsilon E$ is the (i, j)th edge. The weight of each edge $e(i, j)$ is described in Eq. (3.13). The resulting $e(i, j)$ is the negative exponential of the weighted sum squared.

$$e(i,j) = exp\left(-\left(\overline{M(V_i)} - \overline{M(V_j)}\right) W \left(\overline{M(V_i)} - \overline{M(V_j)}\right)^T\right) \tag{3.13}$$

where W is a diagonal matrix of weights. The selection of W determines which metrics are more critical for merging the regions and the selection of W may vary between applications. For example, in an area that light is highly varying with time (not location), we may want to neglect lightness as a metric to associate features, and do so by reducing the weight associated with lightness. In neglecting lightness, we allow shaded objects that are the same to be associated with each other. In another example, let us suppose that we know that we will be in an environment that has color difference over the duration of a mission with different lighting conditions. We may want to neglect color as a matching criteria and use primarily physical characteristics. In both of these scenarios, the metric weights are different. Using normalized cuts (or N-cuts) we select edges that are high enough in magnitude to consider the pair of nodes as one. If we let V_i, V_j denote two nodes, Eq. (3.14) describes the N-cut relationship. The value of the $Ncut(V_i, V_j)$ is the sum of the ratio (the edge being

[7]An affinity matrix represents relationships between nodes - often synonymous with similarity matrix or connectivity matrix

evaluated by the sum of all edges for the node) for both nodes connected by the edge being evaluated.

$$Ncut(V_i, V_j) = \frac{E(V_i, V_j)}{\sum_{\forall i} E(V_i, V_j)} + \frac{E(V_i, V_j)}{\sum_{\forall j} E(V_i, V_j)} \tag{3.14}$$

By normalizing the cuts with the Ncut formulation, we remove the bias for nodes with a large number of edges over other nodes to merge. The criteria for merging two nodes is then:

1. $Ncut(V_i, V_j) > Threshold$
2. $E(V_i, V_j) = \max_{\forall j}(E(V_i, V_j))$
3. $E(V_i, V_j) = \max_{\forall i}(E(V_i, V_j))$

The regions that pass the criteria are merged, E is updated, and the process is repeated until no regions pass the criteria or a max number of iterations are attained. This process is very computationally expensive, but, like the initial segmentation, has the potential to be sped up significantly via parallel computation. Figure (III.9) illustrates the changes in the regions after this process. Table (III.2) shows the tuning parameters in this simulation.

As can be seen in the weighting matrix W, the notebook in the original image, Figure (III.6), is merged together. The background near the tissue box is merged with the background correctly. The bag on the table remains intact.

Table III.1. Sample segmentation: OLT simulation parameters

Tuning Parameter	Simulation Value
W	$[0, 0.2083, 2.0833, 2.0833, 0, 0.0174, 0.1736, 0.1736, 0, 0.0087, 0.0868, 0.0868]10^{-2}$
α	$[-0.1, -0.1, -0.1, -0.1]$
Threshold	0.0036
Max Iterations	10

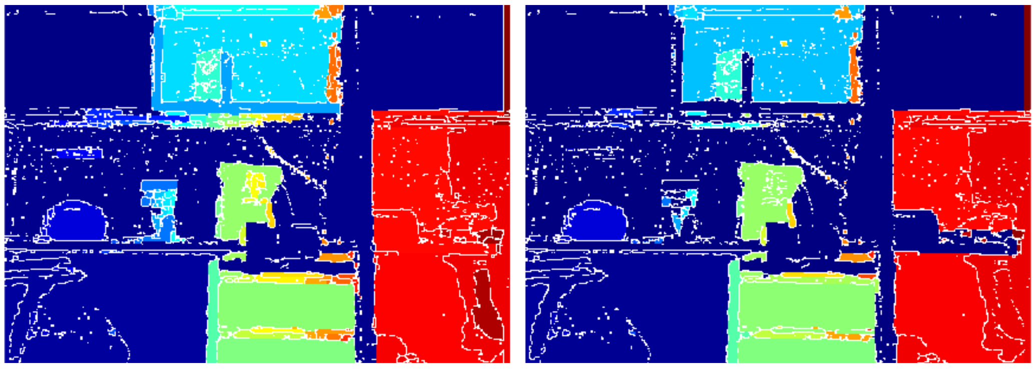


Figure III.9. Sample segmentation: OLT (left) & OLT with N-Cut (right)

III.B.3. K-Means

K-Means [20] is largely classified as a clustering algorithm. It falls in the category with the likes of the Mean-Shift [22] algorithm. Clustering algorithms catergorize data points together by their 'closeness'. Often, closeness is defined by the euclidean distance. K-Means requires a set of 'seed points' to be defined. Often these are randomly selected, as in this study. The seed points represent the 'mean' of a cluster. Given a set of seed points, all points in the data set are assigned to one of these clusters - the one with which its closeness is smallest. After this, the mean generally is different from the seed point. The mean is recalculated and the process is repeated until the clusters do not change. After the steady state is reached, the process is repeated with a new random seeding. This repetition is done some defined number of times. At the end of all the iterations, there is a defined value called the compactness. This is given by Eq. (3.15). Compactness represents the distribution of each grouping from the center. A single repetition may not find the optimal compactness. It is in similar vein to finding a local minimum, but not a global minimum in a Newton gradient descent method of a non-convex function.

$$C = \sum_j \sum_i ||d_{j_i} - m_j||^2 \tag{3.15}$$

The iteration with the lowest compactness is selected for the final clustering. Note that this algorithm is by no means specific to images. This algorithm can be used in any high or low dimensional space of data. As such, the clustering of data may not be adjacent to each other. In other words, two blue balls sitting near each other but not in contact may belong to the same cluster. So these clusters do not

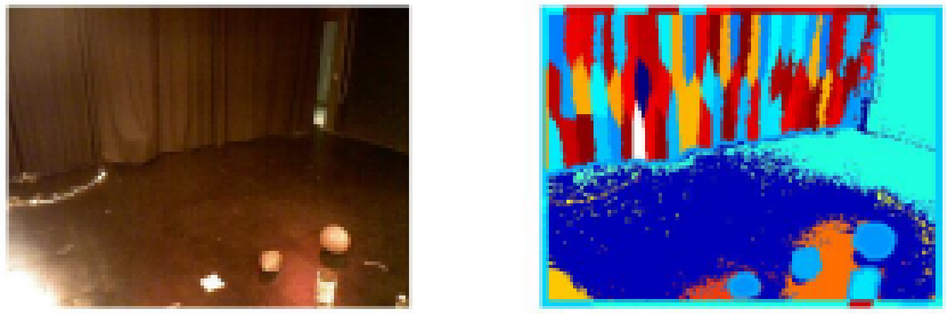


Figure III.10. Sample segmentation: k-means. Left: original color image, right: segmented image. This illustration uses a seed of 50 points and 10 iterations for the best compactness.

represent objects that can be accurately described by closed disconnected geometries. To address this, OpenCV's *connectedcomponents* function is used to identify disconnected components [31]. Figure (III.10) shows a sample of the original color image on the left and the segmented image on the right. Note that the curtain in the backrgound is broken into many segments. The curtain in the image is indistinguishable between each of these segments. The items on the floor are distinctly distinguished however. There are a few segments near the objects that seem to correspond to lighting variations and reflections. This highlights the major challenge in the field of image procesing. Overall, the image segmentation works well; meaning the objects on the floor that are intended to be used as objects in a SLAM process are segmented from the surrounding regions.

III.B.4. Graph

The graph-based method is an extrapolation of the efficient graph-based method developed by Felzenszwalb et. al (2014) [21]. Graph based methods represent the data by a set of vertices and edges. In the context of this method, the vertices

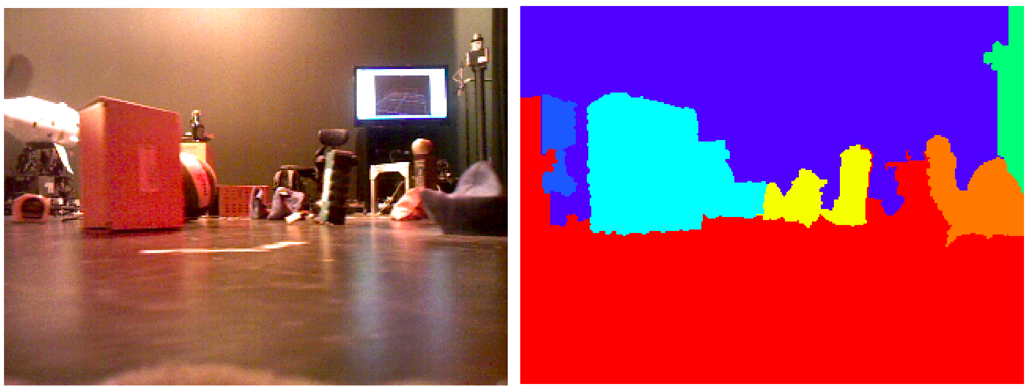


Figure III.11. Sample segmentation: graph-based method. Left: original color image, right: segmented image

represent each pixel and the edges represent the 'difference' with adjacent pixels. Felzenszwalb represents edges by a Euclidean distance between color intensities. This is stated in Eq. (3.16).

$$w((v_i, v_j)) = |I(v_i) - I(v_j)| \tag{3.16}$$

By extrapolating this to an N-dimensional image, we can calculate a weighted Eulidean distance as seen in Eq. (3.17), where M_k is the k^{th} metric.

$$w((v_i, v_j)) = \sum_{k=1}^{N} W(k)|M_k(v_i) - M_k(v_j)| \tag{3.17}$$

The rest of the process introduced by Felzenszwalb remains the same. For a detailed explanation of the process, refer to Felzenszwalb et. al (2014) [21]. Figure (III.11) shows a sample segmentation using this method.

Table III.2. Segmentation evaluatuon: watershed verse OLT

	Number of Segments	Elapsed Time (ms)
Watershed Algorithm	9705	453.029
OLT	167	539.629

III.C. Method Trade Study

It is important to explain why one method was considered obsolete to the other when progressing between each. Here it these trade-offs will be discussed.

In Figure III.7, you can see how the watershed algorithm oversegments the image into many segments. In Figure III.8, OLT segments the image into intuitive pieces. Though still oversegmented, this image far fewer less segments. This reduction greatly reduces the computational cost of region merging algorithms. If the computational power of the system is low, the primary segmentation is still usable for feature extraction. Table (III.2) compares each method.

It is worth noting that overlaid thresholding is approximately 20% slower than the watershed algorithm on a single thread implementation. However, there are approximately 98% less segments. Overlaid thresholding necessitates calculating means and standard deviations of numerous layers of metrics and image multiplication. Both of these processes can be easily made in parallel to improve computation time.

Even at a superficial level, it was easy to distinguish whether watershed or OLT performs better. However, it is not so straight forward with OLT, K-Means, and Graph-based methods. The distinguishing characteristics are blurry. So here, the important characteristics with regard to image segmentation will be discussed. Simply put, they are:

1. Speed,
2. Repeatability, &
3. Uniqueness.

The speed to take a set of data and produce a segmented image must be fast enough to be implemented in real time. At this stage, it is important to not prematurely optimize. As such, the speed of the algorithm should be evaluated with the likelihood for speed increase with parallel processing and other optimization techniques. This makes this first performance characteristic much more qualitative than quantitative. Repeatability means that a segmentation in one frame exists in the next frame (which is nearly identical). An algorithm that is not repeatable does not provide information that can be tracked. Additionally, in concert with repeatability, the segmentations must be unique. The segmentations must segment the image in such a way that unique features are highlighted as different objects. This allows for a more robust data association process in SLAM.

III.C.1. Experiment

III.C.1.a. Objective

In order to qualitatively and quantitatively compare image processing techniques, the key characteristics of the system that are important need to be identified. The characteristics that are important should directly reflect in the success and performance of CANS. These characteristics should be observable, ideally quantitatively. It is not yet clear whether GIS metrics will be benefit image segmentation or if they will affect the algorithms differently. For this reason, two variations of each algorithm (with GIS information and without GIS information) were ran.

III.C.2. Performance Measures

The important characteristics are:

1. Speed
   (a) In optimized framework, should be < 1 sec. That is a very conservative bound.
2. Repeatable Labels of Interest (LOIs)
   (a) Between successive frames, the same objects are segmented and identified as LOIs.
3. Whole LOIs
   (a) The LOIs are not partial elements of a larger object. It is a partial of a large object if it exists inside a predefined whole object.
4. Extracts all LOIs
   (a) Identifies all LOIs expected to be extracted.
   (b) These must be predefined.

III.C.3. Evaluating Performance Measures

The next important point is how to evaluate the performane measures - either qualitatively or quantitatively.

III.C.3.a. Speed

On a single platform, each algorithm was ran on a set of N images. The processing time for each algorithm was recorded. The same set of N images was used for each algorithm.

III.C.3.b. Repeatable LOIs

For a sequence of N images viewing a set of objects throughout, the following test for each algorithm was ran:

1. Run algorithm.
2. Record which objects were extracted.
3. Repeat 2 through the sequence.
    (a) Record whether the same objects from the first image were extracted.

III.C.3.c. Whole LOIs

For a set of N images, the bounds of whole objects were manually defined and each algorithm was tested. If an LOI is part of but does not encompass all of one of the defined bounds, a failure is recorded.

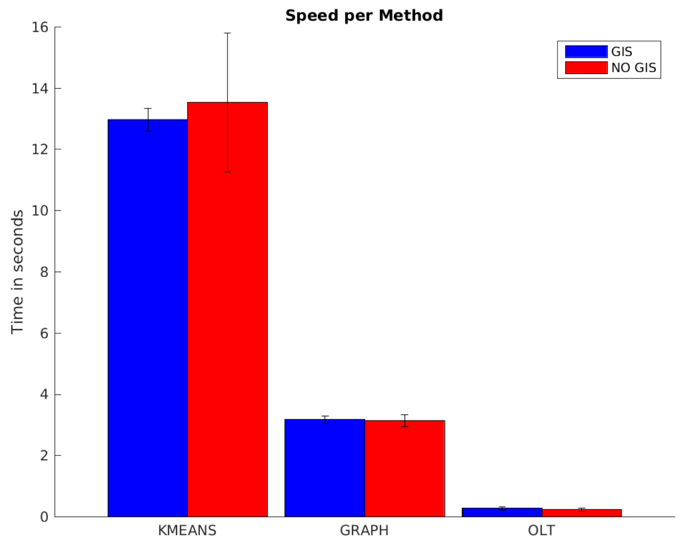


Figure III.12. Speed comparison

III.C.4. Results

III.C.4.a. Speed

Each algorithm (K-Means, Graph, OLT) was run with two variations (with and without GIS information) on 60 images. The algorithms with GIS information used 4 additional metrics. Figure (III.12) shows a bar graph of this test. Note that blue indicates with GIS and red indicates without. The error bars show a one standard deviation from the mean.

The figure shows a clear distinction between K-Means, Graph, and OLT. OLT is the fastest, followed by Graph and then K-Means. The corresponding values are in Table (III.3).

Table III.3. **Speed comparison of image segmentation techniques**

Algorithm	GIS/NO GIS	Mean Time (Seconds)	Standard Deviation Time (Seconds)
K-Means	GIS	12.9762	0.3726
K-Means	NO GIS	13.5384	2.2726
Graph	GIS	3.1741	0.1135
Graph	NO GIS	3.1442	0.1990
OLT	GIS	0.2728	0.0557
OLT	NO GIS	0.2327	0.0611

III.C.4.b. Repeatable LOIs

Given a set of objects that were extracted, how often does that object get highlighted? Using a set of 8 frames collected in succession, each algorithm (K-Means, Graph, & OLT) were run with two variations (with and without GIS information). The GIS metrics used were:

- Slope Azimuth Divergence Index
- Slope Azimuth Convergence Index
- Slope Azimuth Homogeneity Index
- Surface Roughness Factor

Figure (III.13) shows the liklihood that an object is identified if it is identified once in the 8 image sequence. Note that K-Means without GIS information performs the best, followed by Graph without GIS and then K-Means with GIS information. Most notably OLT with GIS information performs the poorest. OLT without GIS

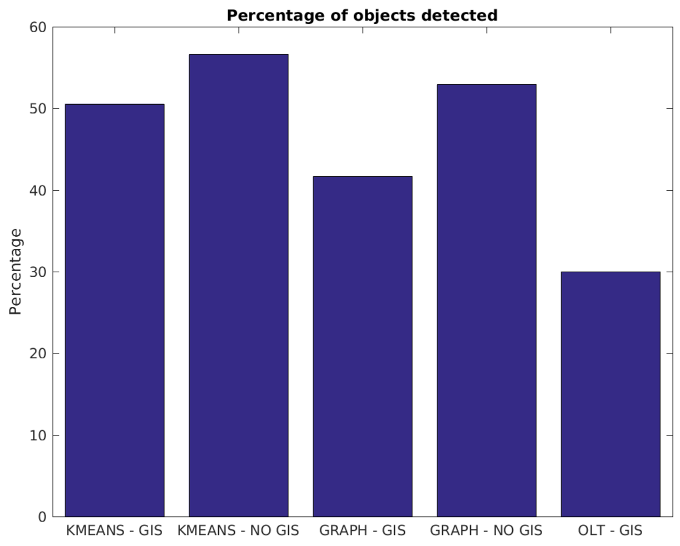


Figure III.13. Repeatability comparison

information was excluded since performance was poor (The images did not resemble any intuitive segmentations and was not consistent between consecutive frames). The corresponding values are listed in Table (III.4).

Figure (III.14) shows the probability distribution of the frequency in which an object is repeatedly observerd. The figure displays a distribution for each algorithm (K-Means, Graph, OLT) with two variation (with and without GIS information). Note that OLT without GIS information is excluded. A more ideal algorithm has a distribution shifted to the right (towards 100%). This would indicate a higher frequency of objects being repeatly identified between successive frames. These distribution curves are fitted to the data. The actual frequency per object, per algorithm is shown in Appendix (VI.B.3).

So what can we say about these results? Figure (III.14) also shows the distribution for each of the bars in bar graph of Figure (III.13). So in addition to what was

Table III.4. Repeatability comparison

Algorithm	GIS/NO GIS	Liklihood of Repeated Object
K-Means	GIS	50.54%
K-Means	NO GIS	56.62%
Graph	GIS	41.67%
Graph	NO GIS	52.94%
OLT	GIS	30.00%

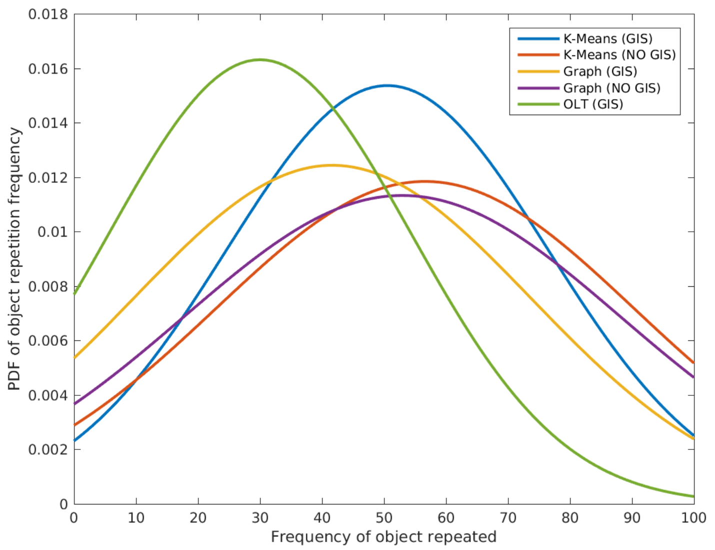


Figure III.14. Repeatability comparison: probability distribution

said about the bar graph, we see that K-Means (GIS), Graph (NO GIS), and Graph (NO GIS) are more widely distributed than K-Means (NO GIS) and OLT (GIS). In addition to the qualitative nature of this test, it was conducted with only one scene and only 8 images; thus, the results should be regarded as suggestive and not conclusive. If given a larger sequence and many more scenes, we could potentially have a different result entirely.

III.C.4.c. Whole LOIs

Say we have extracted some set of objects, what is the impact of each object being represented by several partial regions (or partials) by the image segmentation process? What would happen if in one frame the object is entirely encompassed by a segment, but in the next frame, the object is split into two (or more) separate segments? I propose this is a non-ideal situation. If the proposed scenario occurs, the position (along with all the other metrics) of the object will not accurately represent the whole object. In particular, it will be biased to a particular direction. If this partial object is matched to the whole object in the data association process of SLAM, we introduce an error into the process that is not indicative of the system motion. So, we desire a process that minimizes this as much as possible. Here is presented a set of 8 images, there were 7 objects tracked. That makes for 56 data points. For each data point, it is classsified as a 'Whole segment', 'Partial segment', or not observed at all. Figure (III.15) represents this data in a bar graph. This information is also listed explicitly in Table (III.5).

Here we see that OLT (with GIS information) provides the most percentage of

Table III.5. Whole segmentation comparison

Algorithm	GIS/NO GIS	Percentage Whole	Percentage Partial	Percentage None
K-Means	GIS	25.00%	44.64%	30.36%
K-Means	NO GIS	32.14%	8.93%	58.93%
Graph	GIS	19.64%	16.07%	64.29%
Graph	NO GIS	32.14%	42.86%	25.00%
OLT	GIS	58.93%	5.36%	35.71%

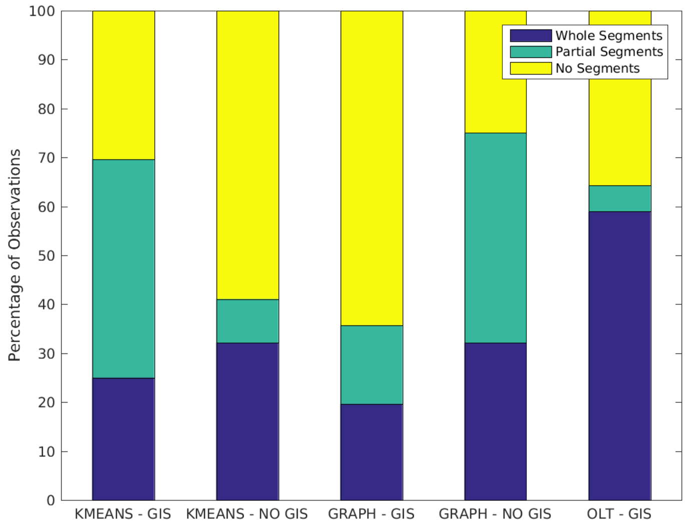


Figure III.15. Whole segmentation comparison

segmentations as wholes and least as partials. This is the best in the performance characteristic. This is followed by Graph (with GIS information). Note that this is better than the other three methods because it minimizes the partial. The logic is: *Get a whole segmentation if possible. If not, don't extract a partial.* This is to minimize the non-representative errors as mentioned previously.

III.C.4.d. Result Interpretation

So evaluating all the qualitative performance characteristics discussed in this section, the following can be said. Speed tests show that K-Means is a significantly slower method than both Graph-based and OLT. OLT shows better speed performance than Graph-based. K-Means (with no GIS information) and Graph-based (with no GIS information) show better repeatability performance. OLT (with GIS information) outperforms all others in ensuring whole segments. So the results do not declare a clear winner, but the Graph-based method seems to perform on par in all regards whereas the others underperform in some regard. For this reason, we will continue a more thorough analysis of the Graph-based method. Also take note that the GIS information tests do not clearly perform consistently better or worse. This is true even within one of the measures. The different algorithms vary whether GIS information is beneficial or not. A strong statement cannot be made on the benefit of GIS information without a more extensive sample space.

III.D. Graph Based Method Investigation

With the version of the graph-based segmentation developed by Felzenszwalb et. al, there were two primary tuning parameters. These are the parameters k and σ. The first serves as an initial thresholding parameter in the beginning of the process and the second serves as a smoothing parameter for gaussian smoothing of the image before processing. With the extrapolation proposed in Eq. (3.17), we introduced N additional tuning parameters - one per metric. This provides a challenge in identifying the best parameters for the task. Ideally, these parameters are chosen methodologically with some basis of reasoning behind them. Due to qualitative characteristics used to evaluate the techniques previously, it would be mindful to perhaps base the methodology, not on empirical observations, but on a fundamental line of reasoning. Here I suggest using a sigmoid function to represent the difference function in Eq. (3.18).

$$w((v_i, v_j)) = \sum_{k=1}^{N} W_k S_k(M_k(v_i), M_k(v_j)) \tag{3.18}$$

where the sigmoid function S_k is defined in Eq. (3.19).

$$S_k = C_k \left(\frac{1}{\left(1 + P_k e^{-\Omega_k \left(x - \frac{L_k}{2}\right)}\right)} - Z_k \right) \tag{3.19}$$

There are a few key properties of this function that are quite convenient. These are:

1. Ranges from 0 to 1.

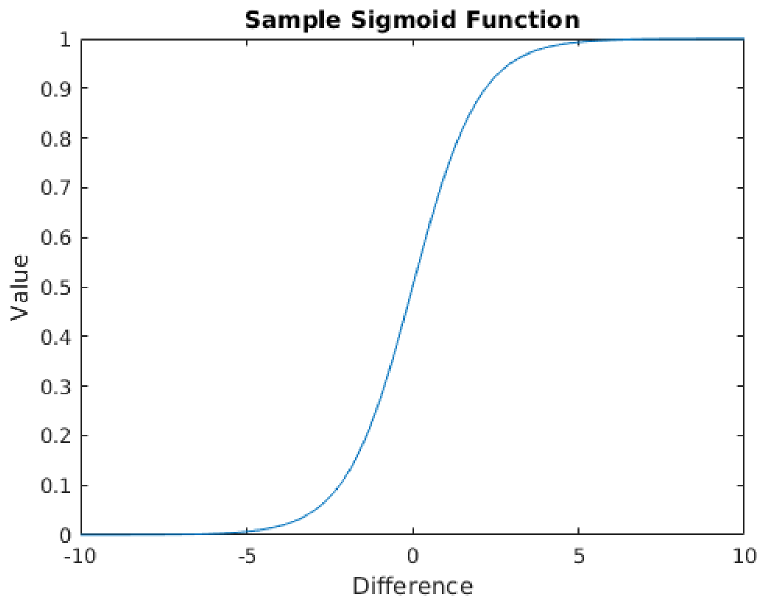


Figure III.16. Simple sigmoid function

2. Nonlinear cap at extremities.

3. Linear in midrange.

4. Solvable for a set of higher level information.

These properties can be seen in the simple sigmoid function in Figure (III.16). Here, it is discussed why each one of these characteristics are important. *1) Ranges from 0 to 1.* By knowing the upper and lower bounds of the differences, we may better determine what the threshold parameter k should be. We know that if the threshold parameter k is larger than N, then any two pixels that are previously unjoined will be joined, regardless of their difference. *2) Nonlinear cap at extremities.* It is very likely that a difference that is moderate may provide increasingly less information as the difference grows. For example, suppose that two pixels with a Z coordinate have a difference of 1 meter. Does 2 meters provide much more infomation about whether the two pixels should be joined? Depending on the application, the answer very well may be no. Similarily, if two adjacent pixels have a Z coordinate difference of 0.01 mm, is that really a difference we can descernably say is different? Or is it possible it may very well be noise? Again, depending on the application, it may be just noise and should be neglected. *3) Linear in midrange.* Though the extremities may not provide linearly increasing amount information, there is likely a midrange that provides this linearly increasing amount of information. The numerical difference in this linear range may correspond to a linearily increasing distinction. *Solvable for a set of higher level information.* Possibly the most important characteristic, we can solve for the parameters of the sigmoid function as a function of higher level information. This information is the noise level, linear region, and saturation level. This forms a system of equations. For example, let's look at a Red intensity sigmoid function. Suppose we conside a difference of 10 as noise, a difference of 150 reaching saturation, and 100 as the midrange. This creates the system of equations in Eqs. (3.20) to (3.23). Here we solve this system of equations with one of MATLAB's numerical solvers [32].

$$0 = C_k \left(\frac{1}{\left(1 + P_k e^{-\Omega_k \left(0 - \frac{2*100}{2}\right)}\right)} - Z_k \right) \tag{3.20}$$

$$0.1 = C_k \left(\frac{1}{\left(1 + P_k e^{-\Omega_k \left(10 - \frac{2*100}{2}\right)}\right)} - Z_k \right) \tag{3.21}$$

$$0.9 = C_k \left(\frac{1}{\left(1 + P_k e^{-\Omega_k \left(150 - \frac{2*100}{2}\right)}\right)} - Z_k \right) \tag{3.22}$$

Table III.6. Sample solved sigmoid parameters

	C (Approx.)	L	P (Approx.)	Z (Approx.)	Ω (Approx.)
Value	1.2668	200	0.4412	0.1718	0.0239

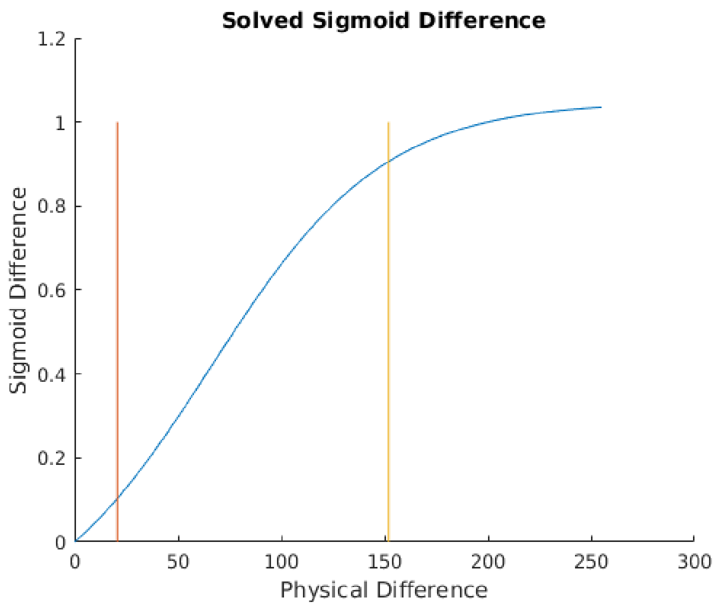


Figure III.17. Sigmoid function example

$$1 = C_k \left(\frac{1}{\left(1 + P_k e^{-\Omega_k \left(2*100 - \frac{2*100}{2}\right)}\right)} - Z_k \right) \tag{3.23}$$

The solution to these equations are in Table (III.6). The sigmoid function that results is depicted in Figure (III.18). Note that the 10% and 90% vertical lines are depicted as well. These are vertical lines that are plotted to intersect with the sigmoid function where the value is at 0.1 (or 10%) and 0.9 (or 90%). This coincides with the noise and saturation levels.

So the next question is what noise levels and saturation levels are appropriate

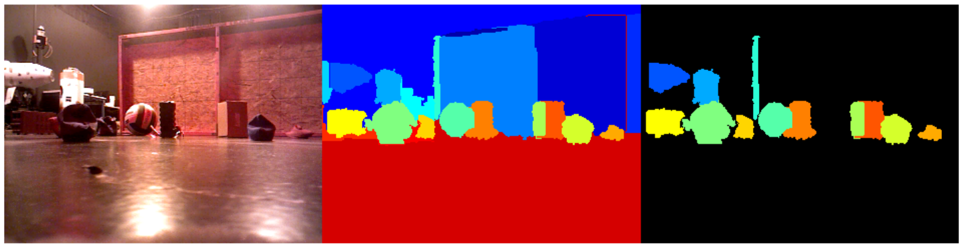


Figure III.18. Sigmoid segmentation example

for each type of information. Here it is reasoned that it depends on application and environment. In the test scenarios that are discussed here, the spatial information only spans approximately 5 meters and the objects being observed are typically about 250 mm in dimension. These are very rough numbers but can give an idea of the scale we care about. So suppose a noise level at 120 mm, saturation level at 400 mm and midrange at 250 mm is set. Similarily, let the color intensity noise level at 30 points, saturation at 150 points and midrange at 90 points. We will also set our graph threshold k at 1 and σ at 0.5. Note that all these parameters can be tuned to the application. Figure (III.18) shows, side-by-side, the original color image, the segmentation, and the regions of the segmentation to be modeled as ellipsoids (left to right).

The segmentation highlights nearly all the major intuitive objects in the field of view. It is important to note that the distinctness of the objects makes this an easier test scenario. Nevertheless, it is a valid scene and one to build from. This segmentation is done with the settings specified by Table (III.7). Here you can see that W is set 1 to allow all channels to be equally weighted. This does not necessarily

Table III.7. Solved sigmoid parameters

Metric	C (Approx.)	L	P (Approx.)	Z (Approx.)	Ω (Approx.)	W
X,Y,Z	1.0638	500	1.2094	0.0245	0.0140	1
R,G,B	1.3876	180	0.7395	0.1893	0.0223	1

have to be the case and a methodology to select this has not been discussed. Choosing this weighting value remains to be a relatively arbitrary selection.

CHAPTER IV

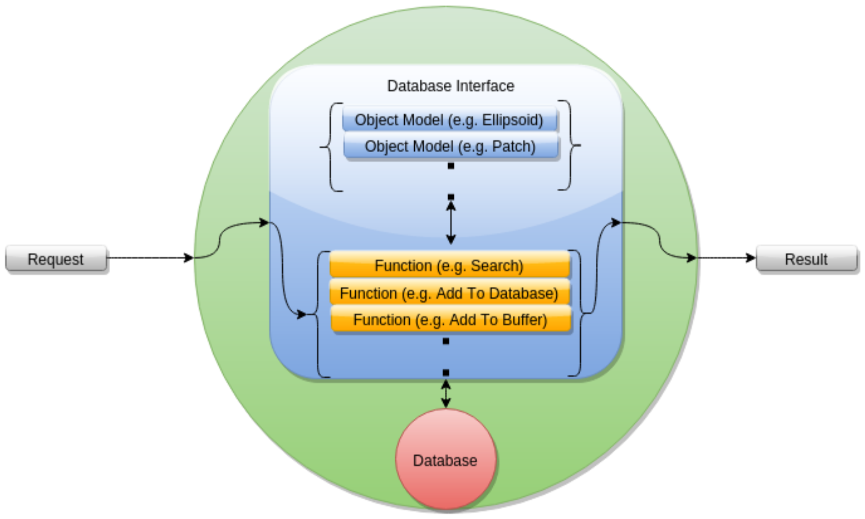
DATA MANAGEMENT

IV.A. Overview

The database management module discussed in Section II is designed around the k-vector search techniques [25, 33]. To illustrate how the database management component is designed, we will provide an overview of the component operation in IV.B followed by a review of k-vector operations in IV.D.

IV.B. Data Management Overview

The database management module can be visualized in Figure IV.1. The database interfaces with a series of functions (shown as orange boxes) that operate in parallel loops. These loops are referred to as agents; agents that exchange information between the database and subsystems (i.e. SLAM, path planning, image processing). As depicted in Figure IV.1, the database lends itself to be utilized by all operations of the system.



Database Interface
Object Model (e.g. Ellipsoid)
Object Model (e.g. Patch)
Request
Function (e.g. Search)
Function (e.g. Add To Database)
Function (e.g. Add To Buffer)
Result
Database

Figure IV.1. Database management overview

IV.C. Functionality Discussion

Some of the agents exchange information with other agents, taking the last produced set of data. For example, we will step through some of the key agents illustrated in Figure (IV.2) through Figure (IV.4). In these flow charts the grey rhombi are data acquired from the subsystems, and the green rhombi are data generated from the functions in one of the interfaces. Note that Metric Weights is a data object that will be taken from the last generated output of another agent not shown. The purpose of this agent is to add features to the buffer - to be later added to the database. As illustrated in Figure (IV.2), the following process is undertaken.

1. *Globalize features*: Globalize the features acquired from the most recent image acquisition by transforming from the body frame into a global reference frame.

2. *Define search range*: Define the search range that will be used to search through the database for features that could be a match. This is a measure that could

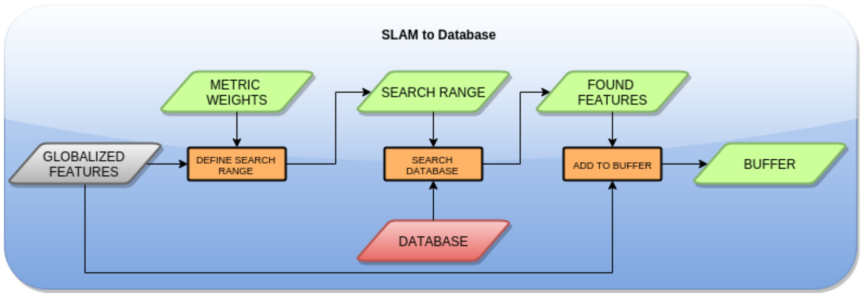


Figure IV.2. SLAM & database interface agent

be variable for different environments and feature descriptions. The metric weights determine the importance of each metric in the search.

3. *Search database*: Search the database in the range provided and select the features that match the features from the image acquisition.

4. *Add to buffer*: Add the matched features and new features to a buffer to update the database on the next update database cycle.

Figure (IV.3) shows the agent in the path planning interface. The purpose of this agent is to grab relevant features from the database for path planning uses. As illustrated in the figure, the following process is undertaken.

1. *Define search range*: Define the search range for features that may pose physical constraints. The objective input is a goal vehicle state (position, attitude and velocity).

2. *Search database*: Search the database through the range given for features and patches within the range in a fashion consistent with the k-vector techniques.

Figure IV.4 shows an agent in the database utility tools. These tools maintain the database and auxiliary data objects that are necessary for the system to operate.

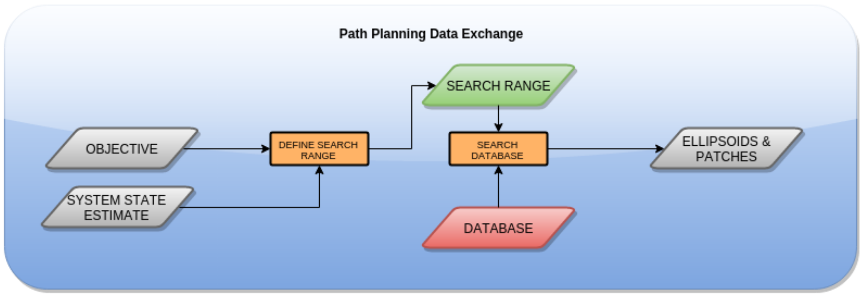


Figure IV.3. Path planning & database interface agent

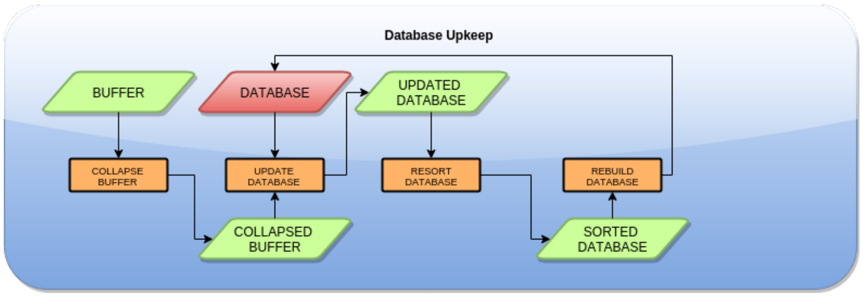


Figure IV.4. Database utility agent - database upkeep

As illustrated in Figure IV.4, the following process is undertaken.

1. *Collapse buffer*: Reduce the buffer by eliminating duplicate features and patches.
2. *Update database*: Update the parameters of features in the database and add new features.
3. *Re-sort database*: Re-sort the sorting indices to reflect the newly updated database in a fashion consistent with the k-vector techniques.
4. *Rebuild database*: Take the updated database (a list consisting of updated features) and reconstruct database to include new and updated features in a fashion consistent with the k-vector techniques.

The multiple agent architecture allows for the different tasks to operate separately, in parallel, at frequencies that are appropriate to that task. Many additional tools can be included in the database management system such as the estimation of dynamic features, adjustment of metric importance to environment conditions, and many more to be developed in future work.

IV.D. K-Vector Overview

Recall that the K-Vector is a searching technique. This technique intends to minimize the time needed (and as a result the number of operations) to find a given item in a list of items. Using such a technique necessitates organizing the data in a special way. The database is formed by two tables. The first table is called the *Object Set*. This table holds the objects (ellipsoids and patches) in the environment. Each row of this table is a separate object. The columns consist of descriptors of the object such as its metrics. For example, a field list is shown in Table IV.1.

This list continues to include Y Centroid, Z Centroid, ellipsoid axis, average color, average GIS indices, and others along with their corresponding uncertainty. The second table is called the *Sorting Indices*. This table holds the sorted Object

Table IV.1. Sample of first 4 columns of object set table

Object ID	Object Type ID	X Centroid	X Centroid Uncertainty	...
1	0	1	0.5	
2	1	4	0.7	

Table IV.2. Sample of first metric in sorting indices table

Sort ID	X Centroid Sort Index	X Centroid K-Vector	...
1	6	0	
2	2	2	
3	3	5	
4	5	5	
5	1	5	
6	4	6	

ID list for each metric and the corresponding K-Vector. For example, an excerpt of a possible table is shown in Table (IV.2). The majroity of the operations reside in this table.

Similar to the Object Set table, this list continues to include sorted Object ID sets for Y Centroid, Z Centroid, ellipsoid axis, average color, average GIS indices, and others along with their corresponding k-vector. The *sort* operation sorts the object ID's by metric value. Each metric has a column (such as 'X Centroid Sort Index') where the object ID's are sorted by that metric (such as 'X Centroid'). *Build* creates the k-vector. Each metric has a column (such as 'X Centroid K-Vector') where the k-vector for that metric (such as 'X Centroid') resides. The k-vector is constructed to hold the indices of a sorted set such that solving Eqs. (4.1) & (4.2) will give the upper and lower search bound indices in the sorted set for search bounds $[a, b]$. *Build* also creates the polynomial cofficients, p & q, for each metric as seen in Eqs. (4.3) and (4.4).

$$k_{start} = k(j_a) + 1 \tag{4.1}$$

$$k_{end} = k(j_b) \tag{4.2}$$

where j_a and j_b are defined by Eq. (4.3) and Eq. (4.4), respectively.

$$j_a = \left\lfloor \frac{a-q}{p} \right\rfloor \tag{4.3}$$

$$j_b = \left\lceil \frac{b-q}{p} \right\rceil \tag{4.4}$$

The construction of the k-vector and a more thorough explanation of the theory can be found in Mortari et al. [25]. Figure IV.5 illustrates a typical search operation. The following steps walks through the diagram.

1. The search bounds $[7, 11]$ are provided.
2. By Eqs. (4.1) & (4.2), the index of the sorted set is calculated.
    - This is illustrated by the blue arrows in Figure IV.5.
3. The object IDs between k_{start} and k_{end} are selected.
    - This is illustrated by the red oval Figure IV.5.
4. The objects that are indicated by the selected Object IDs are grabbed.
    - This is illustrated by the red circles and purple arrows Figure IV.5.

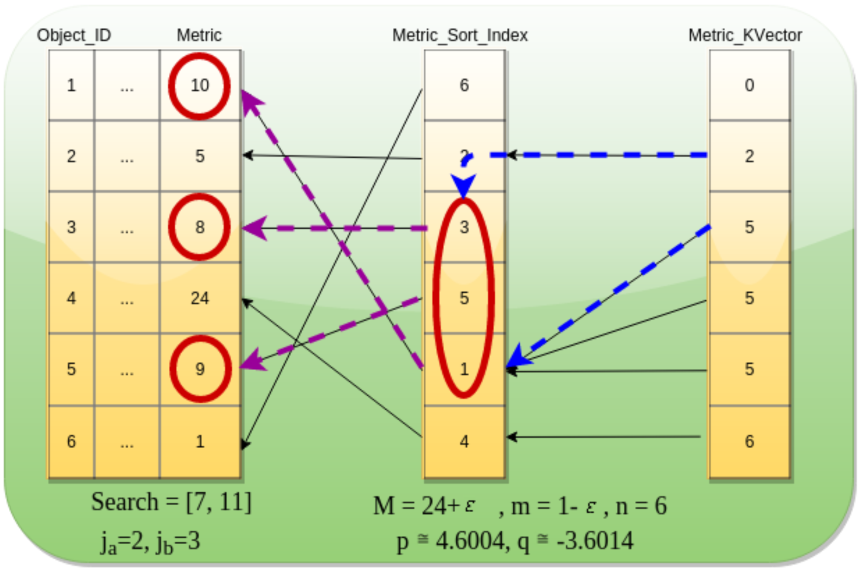


Figure IV.5. Database k-vector relationship illustration

This operation provides a set of matching objects for each metric. In practice the bounds are defined by the uncertainty of the observed object formed in image processing and a set intersection algorithm is performed on the set for each metric to find the object(s) satisfying all bounds. If multiple matches are found, a trimming routine is employed to find the match with the closest Euclidean distance to the observed feature. Using the k-vector to perform these operations is efficient because the position of search bounds in a sorted set is explicitly calculated by the solution of a linear equation (Eqs. (4.1) to (4.4)). This is in contrast to comparing the values of each element of the sorted set until the bounds are found.

IV.E. In-Depth Investigation

In order to qualitatively and quantitatively evaluate the database management subsystem, we need to identify key characteristics of the system that are important.

The characteristics that are important should be directly reflected in the success and performance of CANS. These characteristics should observable, ideally quantitatively.

IV.E.1. Performance Measures

Here is a list of characteristics that impacts the perfomance of the overall system.

1. Speed

    (a) Given a set of N database objects and Z metrics,

        i. How fast is *define search range*?

        ii. How fast is *search*?

        iii. How fast is *sort*?

        iv. How fast is *build*?

2. Accuracy

    (a) Given N objects and bounds to include one object,

    (b) How often is the correct object included?

    (c) How many additional objects are included?

IV.E.2. Evaluating Performance Measures

To evaluate the characteristics, the following steps were repeated, measuring the time elapsed at each point until the database had 1000 objects.

1. Define Search Range for object in previous addition.

   (a) Except in the first step of the test.

2. Search database with calculated search range.
3. Trim result set to one object.
4. Add simulated found features and new features to buffer.
5. Add buffer to database.
6. Update found features in database.
7. Sort database.
8. Build database.

IV.E.3. Expectations

Generally, we expect the entire loop to decrease in speed as the number of objects (N) in the database increases. By inspection[1], We expect the time complexity of *Define Search Range* to follow Eq. (4.5).

$$\propto NZ \tag{4.5}$$

We expect *Search* to follow Eq. (4.6).

$$\propto (Z-1)\,(4\sigma_{MAX}-1) \tag{4.6}$$

[1]Inspection here is closely related to big O analysis, but retains some lower order terms.

where σ_{MAX} is the maximum uncertainty of a metric in search object. We expect *Trim* to follow Eq. (4.7).

$$\propto \sigma_{MAX} \tag{4.7}$$

We expect *Add to Buffer*, *Add to Database*, & *Update* to be independent of N and Z. We expect *Sort* to follow Eq. (4.8).

$$\propto ZN\log_2(N) \tag{4.8}$$

We expect *Build* to follow Eq. (4.9).

$$\propto (BZ)\,N^2 + \left(AZ - \frac{1}{2}BZ\right)N + (D - AZ) \tag{4.9}$$

where A, B, D are constants.

IV.E.4. Results

Figure (IV.6) shows all the timed operations as function of the number of objects in the database. Note that 'Loop' is the total time elapsed for an entire loop, or steps 1-8 in the above section. So this is approximately the sum of all the other operations. Note that *sort* and *build* take majority of the elapsed time. So much so that the other operations are not that visible in the figure. The K-Vector is designed for static operations where *sort* & *build* are not frequently performed; so, it is expected that *sort* and *build* are the primary culprits of undermining the performance. Let's return to this point shortly.

Figure (IV.7) shows the previously difficult-to-see operations. Note that all the

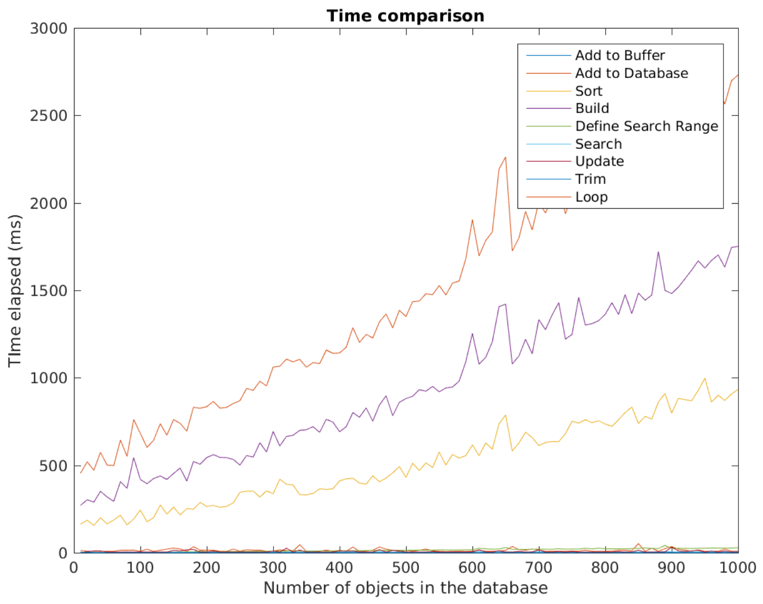


Figure IV.6. All operations' elapsed time as a function of objects in the database

operations are primarily constant, aside from the frequent spike, except the *Define Search Range* operation. This matches our predictions from the previous section. For individual plots of each operations with its corresponding fit curve (fit abides by the predicted form), see Appendix (VI.B.3). Table (IV.3) show the RMSE and the normalized RMSE (NRMSE) for each of operations with respected to the fitted line. Here NRSME is defined to by Eq. (4.10).

$$NRMSE = \sqrt{\frac{1}{l}\sum_{i=1}^{l}\left(\frac{Y_i - D_i}{D_i}\right)} \tag{4.10}$$

where D is the data, Y is the fitted function, and l is the length of the data set.

Note that the *build* operation has the largest deviation, followed by *sort*. These are also the functions of largest value. NRMSE shows that *build* & *sort* are not the

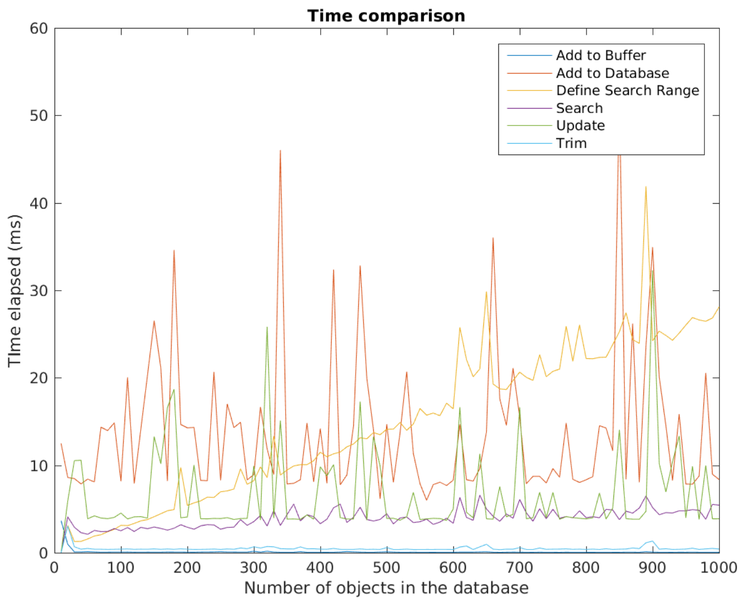


Figure IV.7. Faster operations' elapsed time as a function of objects in the database

largest in this measure.[2] Generally, these measures show that the predictions are accruate.

[2]NRMSE gives a normalized error to offset any large variances due to the scale of larger measurements. In essence, this captures the deviation as a ratio with the magnitude of the measurement.

Table IV.3. RSME values for database operations; (N)RMSE with respect to prediction from fit curves.

Operations	RMSE (ms)	NRMSE (ms)
Add to Buffer	0.3685	0.9573
Add to Database	8.3897	0.5481
Sort	40.8561	0.0877
Build	70.2219	0.0735
Define Search Range	2.4756	0.1329
Search	1.0536	0.2752
Update	5.0257	0.6078
Trim	0.3025	0.2536

IV.F. Dynamic K-Vector Study

K-Vector was originally designed to be used with a static database. As was seen in Section (IV.E), the sort and build operations take, by far, the most time. As suggested by Mortari et. al [33], there may be ways to create a K-Vector for a dynamic database. Here we investigate a method for going about this dynamic K-Vector. When the K-Vector is used for operations, we are solving the linear Eq (4.11).

$$x = \frac{y-b}{m} \tag{4.11}$$

Building will create new m and b constants for each metric. Suppose that we choose not to build and sort and instead just insert appropriate values where they belong. The solution to the polynomial in Eq. (4.11) will become increasingly less accurate. To see how innacurate, let's review how m and b are calculated in Eqs. (4.12) & (4.13).

$$m = \frac{y_{max} - y_{min} + 2\zeta}{n - 1} \tag{4.12}$$

$$b = y_{min} - m - \zeta \tag{4.13}$$

where n is the nummber of objects in the set, ζ is some small number, y_{max} is the maximum value of the set, and y_{min} is the minimum value of the set. If we assume that the objects coming in are not outside the extremities (i.e.$< y_{max}$ and $> y_{min}$), then m becomes increasingly less accurate as n grows. To determine how much, see that the index $A := 1 - m_{OLD}/m_{NEW}$ is defined in Eq. (4.14).

$$A = 1 - \frac{m_{OLD}}{m_{NEW}} = \frac{n_B}{1 - n_{OLD}} \tag{4.14}$$

where n_B is the number of objects added since the last static build, n_{OLD} are the number of objects in the database at the last build, m_{OLD} is the m coefficient calculated from the last static build, and m_{NEW} is what the coefficient should be. By using this as a criteria for when the 'Allowance' has become to large (and thus too innacurate), one can choose to do a static sort and build to reset A to 0. Let it be stressed that this Allowance is calculated per metric and only the metric that surpassed its Allowance will need to perform a static *sort*, *build*, & *add to database*. Now, that we know when we are going to do a dynamic sort and build, let's disucss exactly what this entails. This operation, as performed here, combines the functionality of sort, build, and add to database.

1. Acquire feature buffer to be added.
2. If $A < A_{THRESH}$, continue. If not:
   (a) Revert to static *sort*, *build*, & *add to database*.
   (b) Exit sequence
3. Solve Eq. (4.11) for index, x, in K-Vector for each feature.
4. Attain the K-Vector values at index (x) of the K-Vector.
   (a) Compare with feature at index.
   (b) If buffer feature metric is less-than database feature metric, decrease index.
   (c) Repeat step (4b).
5. Compare values in sorted object ID set for proper placement of new feature.
6. Insert object ID.
7. Repeat through entire set.
8. Calculate A from Eq. (4.14).

To evaluate the performance of this method, a random object set of 1000 was generated with a fixed uncertainty. A similar sequence to that followed in Section (IV.E) was followed in these tests. The steps were revised to the following:

1. Define Search Range for object in previous addition.

    (a) Except first step of the test.

2. Search database with calculated search range.

3. Trim result set to one object.

4. Add found object and new objects to buffer.

5. Add buffer to database.

6. Run Dynamic Add to database.

    (a) Follows steps described above.

7. Record allowance thresholds surpassed and extremities surpassed.

Note that here *Update* is not used. The current state of the dynamic k-vector operations is not yet setup to run *Update*. The other functionalities are sufficient to demonstrate preliminary results on the benefit of a dynamic K-Vector. To do an update, the K-Vector indices and sorting indices need to be relocated and the indices in between the old and new location will be shifted. Additionally, we are recording the allowance thresholds surpassed and the extremities (y_{max}, y_{min}) surpassed since both mean a static sort, build, and add to database are run for the metric that caused the flag. Figure (IV.8) shows the dynamic K-Vector operational loop elapsed time for an allowance ranging 0 to 1. Note that there is no noticeable difference.

Figure (IV.9) shows the difference with respect to an Allowance of 0 (static sort, build, add to database everytime) fo each of the allowance threshold settings.

Here it can clearly be seen there is no performance improvement despite the fact that there are fewer static sorts and builds. This can be seen in Figure (IV.10). Recall

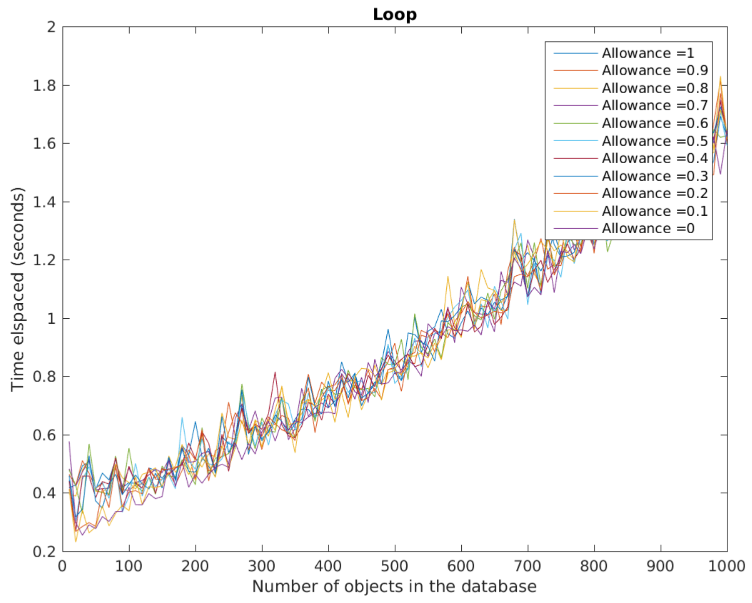


Figure IV.8. Loop time elapsed for dynamic k-vector ranging allowance 0 to 1

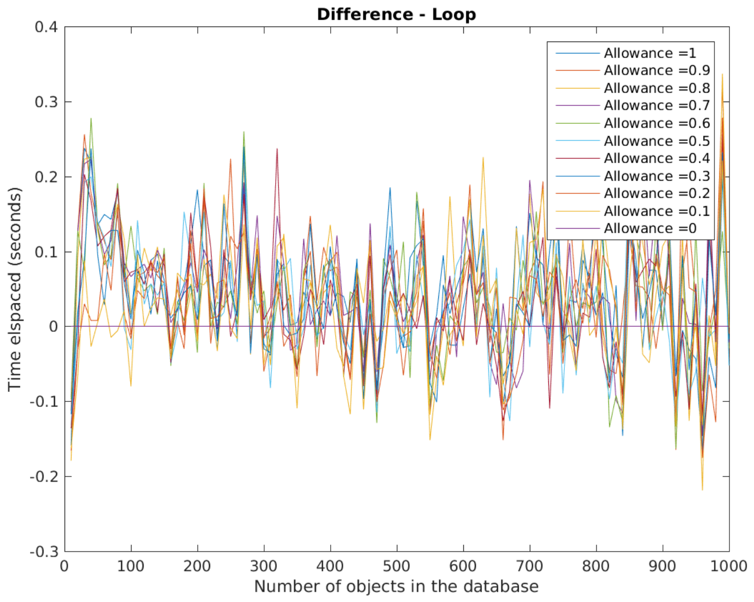


Figure IV.9. Loop time elapsed difference - with respect to allowance of 0 - for dynamic k-vector ranging allowance 0 to 1

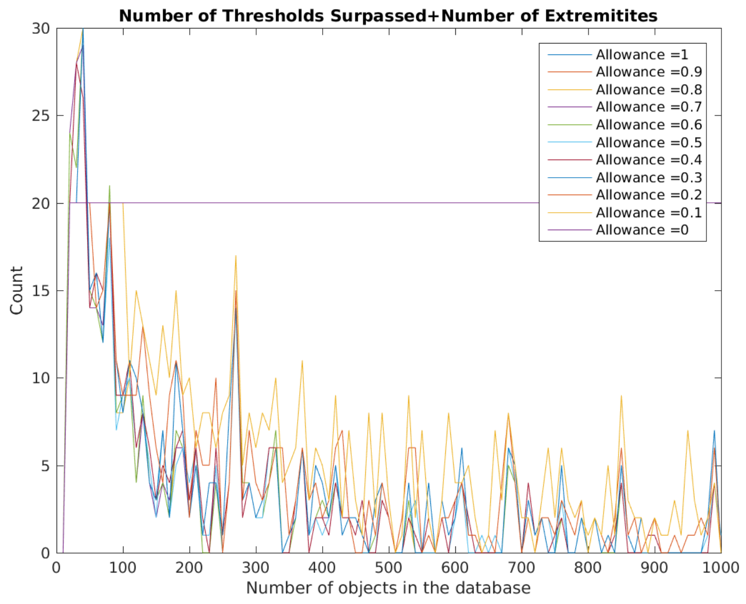


Figure IV.10. Thresholds and extremeties surpassed, suggesting number of static builds.

that, static builds and sorts only occur when that metric has a threshold surpassed or extremity surpassed. So the count of these should scale with the number of static sorts and builds. Figure (IV.10) then indicates that the static sorts and builds are in fact decreasing despite minimal change in the time elapsed for the operation.

It is suspected that this is due to a large amount of disk operations that occur in the current implementation. The disk operations occur at every instance of the operational functions. In particular, sort and build operate on large amounts of the data when shifting indices in the table. Appendix (VI.B.3) shows figures for all the individual operations, corresponding difference plots, and measures of search success. Though increased performance is not visible here, it is expected that this approach will show significant improvement in a more appropriate implementation in RAM. Discussion of steps to investigate this further will be discussed in Section (VI.B).

CHAPTER V

CANS PERFORMANCE

As mentioned at the beginning of this thesis, the overarching goal of this work is a Cohesive Autonomous Navigation System (CANS). Recall how the system is formed as depicted in Figure (II.1) in Section (II). In this section, the overall performance of the system will be characterized. This will include work from the other modules in addition to image processing and data management. Let's start by recalling the objective of CANS. The desire is to have an autonomous navigation system that represents information in a way that is helpful to SLAM and path planning. Here, autonomous navigation means that the system can localize, map the environment, plan trajectories, and determine mission objectives. We also wish to use higher level information, such as GIS metrics, to help with image segmentation and data association. Excluding path planning in loop, we show results of the combined system.

The settings used in these results are listed in Table (V.1). The uncertainties listed under 'Data Management' are the values mentioned at the end of Section (IV.D). The uncertainties listed under 'SLAM' are the values used to form the Q and R matrix for the Unscented Kalman Filter used in the SLAM process (for process model and observation model, respectively). For more information on the details of a UKF refer to Wan et al [34]. Note that the uncertainty for attitude is listed as a

[1]Control is generated by estimating the acceleration (angular or translational) through forward finite element differencing of the state trajectories recorded by the VICON system.

Table V.1. CANS simulation settings

Setting	Value
Image Processing	-
Segmentation Method	Graph - Sigmoid
Metrics	X,Y,Z,R,G,B
W	$[1, 1, 1, 1, 1, 1]$
P	$\approx [1.20941, 1.20941, 1.20941, 2.84293, 2.84293, 2.84293]$
Ω	$\approx [0.01397, 0.01397, 0.01397, 0.0503, 0.0503, 0.0503]$
C	$\approx [1.06384, 1.06384, 1.06384, 1.08637, 1.08637, 1.08637]$
Z	$\approx [0.02453, 0.02453, 0.02453, 0.04980, 0.04980, 0.04980]$
L	$[500, 500, 500, 180, 180, 180]$
σ	0.5
k	1
Predicate: Min Pixel Count	230
Predicate: Max Pixel Count	8000
Predicate: Invalid Pixel Count	25%
Predicate: Max Distance from sensor	2 meters
Predicate: Min Distance from sensor	0.2 meters
Point Exclusion	25%
Post-Extraction Predicate: Max Aspect Ratio	13
Post-Extraction Predicate: Max Axis Magnitude	0.3 meters
Post-Extraction Predicate: Min Axis Magnitude	0.001 meters
Post-Extraction Predicate: Min Largest Axis Magnitude	0.02 meters
Data Management	-
X,Y,Z Centroid Uncertainty	0.1
Axis Magnitude Uncertainty	0.25
Quaternion Element Uncertainty	2
RGB Uncertainty	130
GIS Uncertainty	130
SLAM	-
Process Model X,Y,Z Uncertainty	$4.1 \cdot 10^{-8}$
Process Model V_X,V_Y,V_Z Uncertainty	$[1.2, 1.2, 1.225 \cdot 10^{-3}] \cdot 10^{-3}$
Process Model θ_1,θ_2,θ_3 Uncertainty	$3.6 \cdot 10^{-6}$
Process Model ω_1,ω,ω_3 Uncertainty	$1.6 \cdot 10^{-1}$
Observation X,Y,Z Uncertainty	$4.1 \cdot 10^{-3}$
Observation Axis Magnitude Uncertainty	$[6.4, 6.4, 6.4 \cdot 10^{-3}] \cdot 10^{-4}$
Observation θ_1,θ_2,θ_3 Uncertainty	$3.6 \cdot 10^{-7}$
Observation ω_1,ω_2,ω_3 Uncertainty	$6.4 \cdot 10^{-5}$
Control[1]	Attitude Control Only

three parameter set. However, the features and system attitude are represented with a quaternion set. To avoid the nonlinearities of quaternions, the UKF uses a rotation vector representation. For more details on the implementation of the quaternions in a UKF refer to Kraft et al [35].

The following sections (V.A) and (V.B) will show results for SLAM and path planning, respectively. These results are built off of the previous sections discussing image processing and data management. The observations made for SLAM are the outputs of the image processing module. SLAM stores these observations, represented in a global frame in the database. SLAM and path planning use the objects stored in the database via data management tools for data association (SLAM) and obstruction representation (path planning). All of these functions are derived from the outputs of the image processing module.

V.A. SLAM

Figure (V.1) shows the system following a path. Note that the blue path is the truth path as recorded by a VICON system. The red path is the SLAM estimated path. The arrows on the trajectories show the orientation of the system (blue - according to VICON, red - according to SLAM estimate). The square icons in the figure represent the centroids of the ellipsoids stored in the database. At first glance, the system does perform the required task with some level of noise. The system seems to loose track (and recover) around the corners. In particular, the first and second corners (the system starts on the right hand side and travels counter-clockwise). For

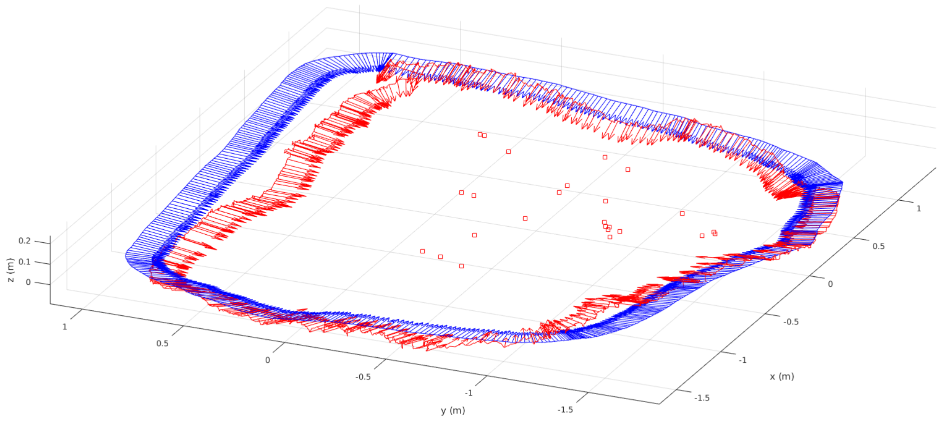


Figure V.1. System state time histories

a more interactive demonstration of the system, see the corresponding animation[1].

Figures (V.2) & (V.3) show the time histories of spatial and angular state time histories, respectively. Here, the discrepancy between the truth and estimate is illustrated. As in Figure (V.1), the red plot indicates the SLAM estimate and the blue plot indicates the VICON truth. The resulting RMSE values are listed in Table (V.2). As can be seen in Figure (V.2) and Table (V.2), the Z direction has the smallest spatial RMSE. Figure (V.2) shows the deviation from the path at the bends. It can be seen in the 3D plot in Figure (V.1) as well. The SLAM estimate of the translational velocities is a much smoother curve than the VICON information. The body rotational rates matches very closely with the noisy VICON data. This should be expected since this demonstration uses attitude control derived from the VICON attitude track. Figure (V.4) shows the error explicitly per time step.

[1]Path animation. URL: `https://www.youtube.com/watch?v=wH86OTOjB7A`

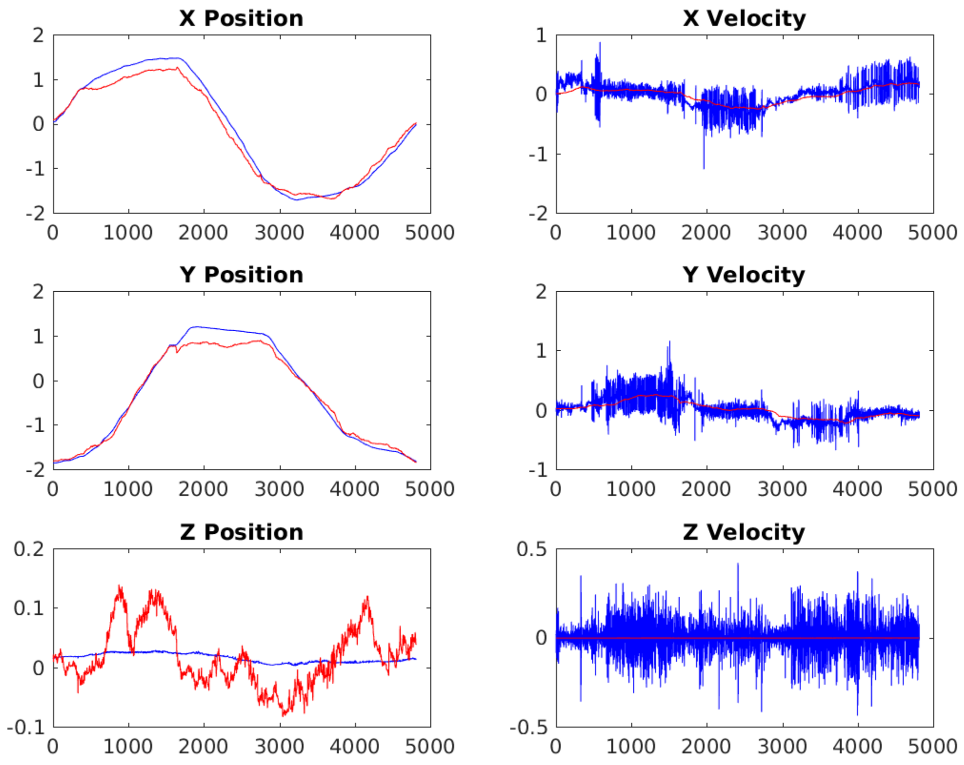


Figure V.2. System spatial state time histories: blue - truth, red - SLAM estimate

Table V.2. CANS RMSE for state time histories

X (m)	Y (m)	Z (m)	V_X m/s	V_Y m/s	V_Z m/s	ϕ (degrees)
0.1829	0.1653	0.0466	0.1187	0.1186	0.0793	12.1210

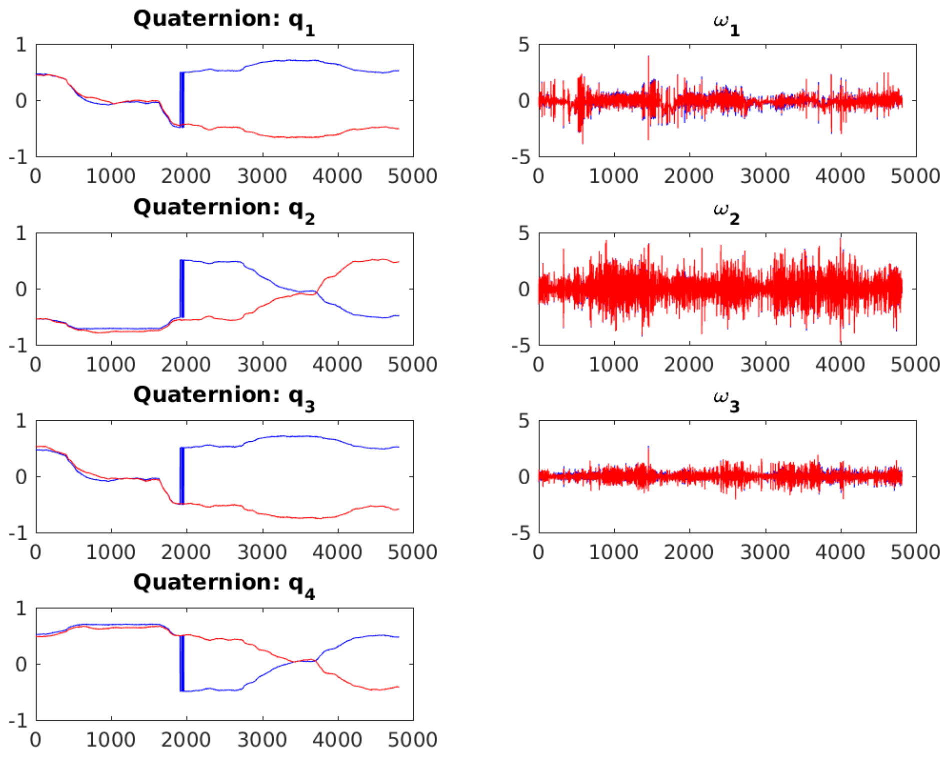


Figure V.3. System angular state time histories. blue - truth, red - SLAM estimate

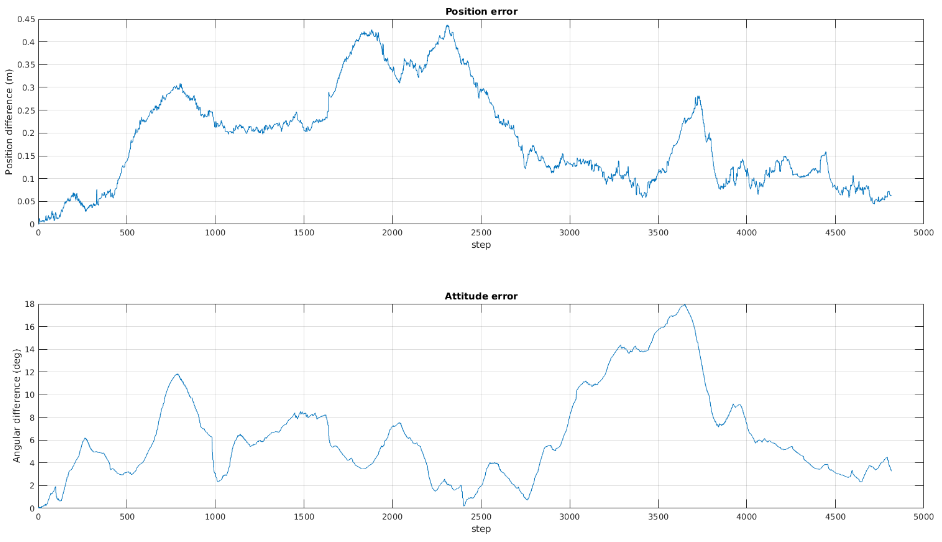


Figure V.4. Error time histories

This demonstration illustrates that the system successfully maps its environment well enough to estimates its own location, but the computations were performed off-line. The next stage is to reach real-time operational capability. To this end, the timing of CANS was recorded. In each time step, CANS may or may not make an observation. CANS was run with a 100 Hz data set with an observation (image acquisition) at 30 Hz. So a time step makes an observation about every 3 time steps. The image processing module contributes to the time elapsed during those steps. Additionally, data association agent is ran in observation time steps- as seen in Figure (IV.2) in Section (IV.C). Figure (V.5) shows the time elapsed in three scenarios: 1) observation time steps, 2) observation without image processing elapsed time, and 3) the non-observation time steps. This figure is very instructive because it shows where computation speed can be improved. Image processing and database operations are the primary contributors to the time elapsed in each observation step.

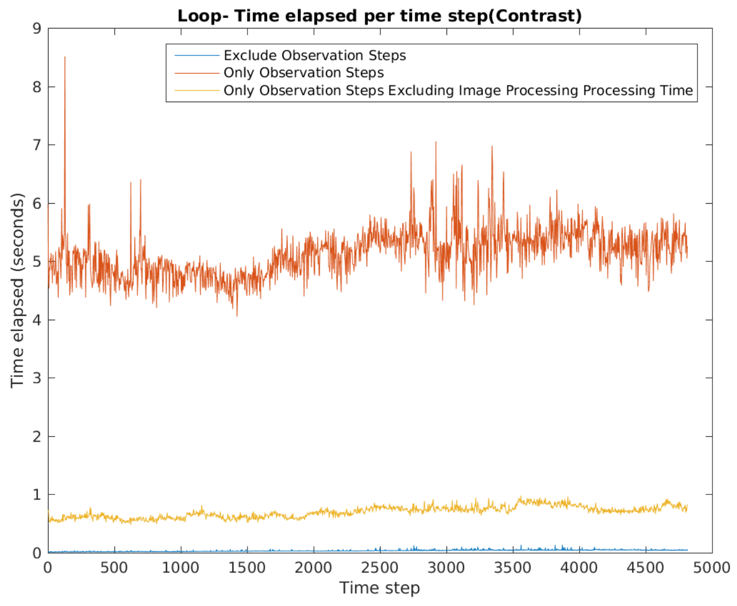


Figure V.5. Time elapsed per time step

Many more interesting parameters as a function of time are in Appendix (VI.B.3).

To further illustrate where speed performance most likely lies, Figure (V.6) shows the overview of CANS with the respective time elapsed in a given time step. Note that the path planning module is excluded since it was not run concurrently in this demonstration. This time step is the next observation step after the half-way point through the SLAM path shown in Figure (V.1). There are 60 objects in the database. It can be seen that *Image Processing* takes the most time by a large margin. The *Image Processing* module's similar illustration is shown in Figure (V.7). Here, note that the *Patch Extraction* is not shown as it is in Figure (III.1), Section (III.A). As mentioned in that section, patches are still work in progress and thus are not presented in the timing study. The sum of the times in Figure (V.7) do not equal to the corresponding time elaspded in Figure (V.6). There is some additional overhead time consumption. The time elapsed for the *Image Pipeline* block is comprised of constructing strings for reading a *.mat*[2] file. In a real-time system, data would be read from a sensor such as the Xtion Pro - where this *.mat* file originated. Though more distributed than Figure (V.6), the bulk of the time elapsed occurs in the *Segment* block. Figure (V.8) shows the steps taken between the *SLAM* module and the *Data Management* module. The grey block groups the two filters together for the timing block indicated. The data management operations are the blocks in green. These two blocks take more time than the operations by SLAM. Similarily to the *Image Processing* module, the sum will not add up to the overall time consumption. Figures (V.6) to (V.8) show good estimates of the timing of their corresponding operations. These estimates are rounded to the nearest 1 millisecond.[3]

[2]MATLAB's data storage structure.

[3]Recall all timing simulations were conducted on the system with the specifications listed in Table (I.1).

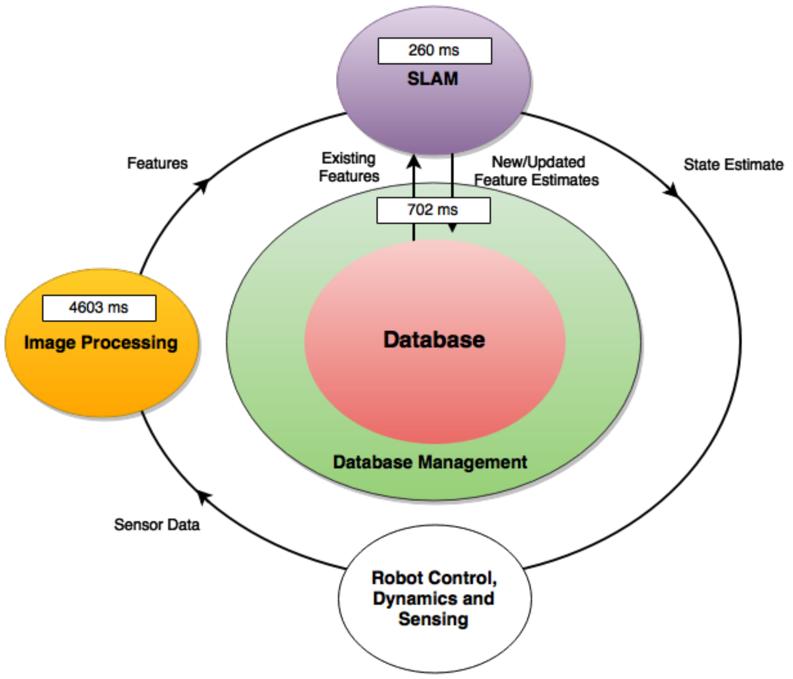


Figure V.6. CANS overview with elapsed time: elasped times for each module are listed in the white boxes.

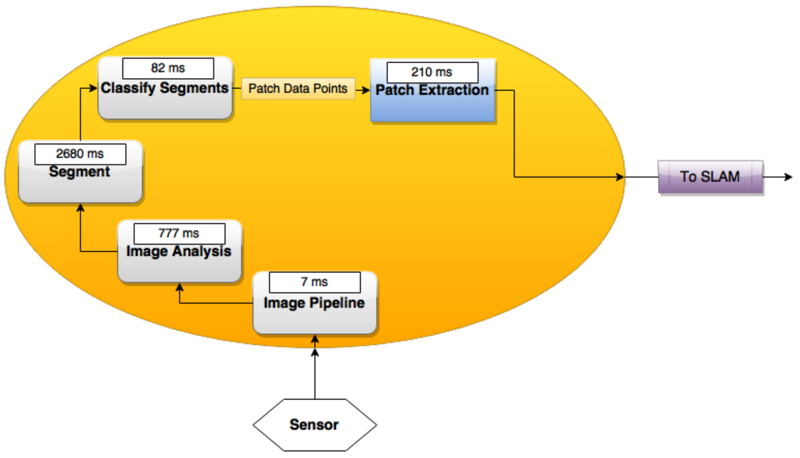


Figure V.7. Image processing overview with elapsed time: elasped times for each module are listed in the white boxes.

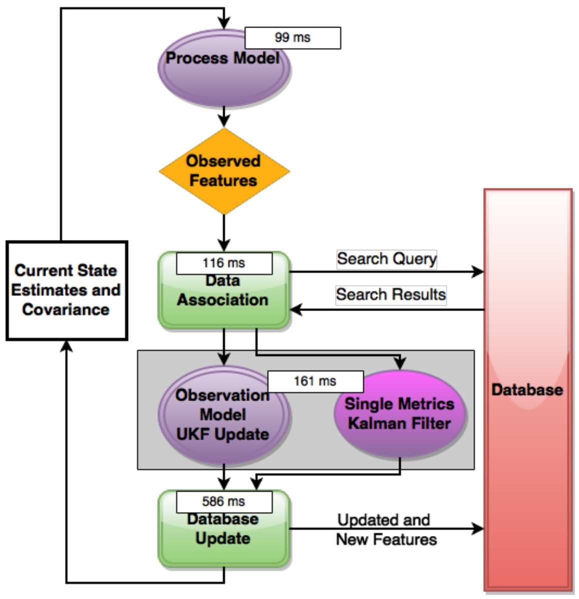


Figure V.8. SLAM overview with elapsed time: elasped times for each module are listed in the white boxes.

V.B. Path Planning

The path planning algorithm is not the primary focus of this work, but it is claimed that CANS allows a path planning algorithm to use the same data used by SLAM. In this work, the ASTRO [12] path planning algorithm is used. This algorithm plans a path from an intitial state to a final state and iteratively replans until an optimal solution is found. In each iteration, a prism around the previous path is formed to search the database for obstacles. This search follows the agent described in Section (IV.C), Figure (IV.3). Figure (V.9) shows a planned path through the dataset generated by the demonstration in Section (V.A). The algorithm planned a trajectory from $(-0.45, -1.3, 0.02)$ to $(0.2, 0.25, 0.05)$ with zero initial and final velocities. The blue ellipsoids represent the ellipsoids attained through the search. All ellispoids that exist in the database are displayed. Note the scene is awfully

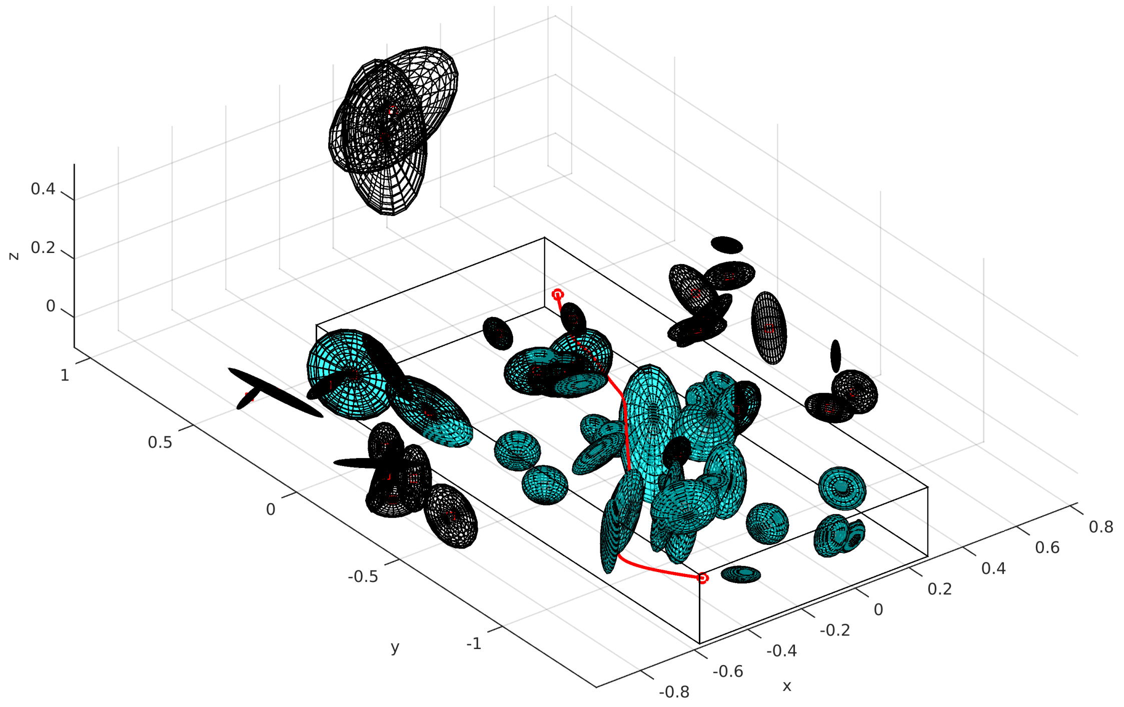


Figure V.9. ASTRO trajectory through demonstration dataset

cluttered (60 ellispoids) though there less than 30 distinct objects in the scene. This will be discussed in Section (VI.B).

CHAPTER VI

CONCLUSIONS & FUTURE WORK

VI.A. Conclusions

In this section, conclusions about the image processing and data management are discussed. In addition, the entire system is discussed. Here, we wish to make statements about the current state of the system and what further extrapolations we can make from here. Naturally, a discussion of future work will follow this section.

VI.A.1. Image Processing

The image processing module was shown to be operating at a sufficient level. The raw data was refomatted through the pipeline, produced higher level information such as the GIS metrics, segmented consistently between frames, and was able to extract elliposid representations of the objects in the frame. The algorithm chosen with a series of relatively subjective and qualitative tests. The graph-based segmentation method was selected. This graph-based method was investigated further for potential means to improve performance. Different difference functions were investigated to some level of success. There is still some ambiguity in tuning parameter selection. It is still not clear whether GIS information was beneficial in image segmentation or data association. The image processing module used in end-to-end system demonsration used X, Y, Z, R, G, & B metrics for segmentation. Though many more (including GIS information) were calculated and used in data association. More work is to be done before making a strong statement about the benefit of GIS information.

VI.A.2. Data Management

The data management structure was explained. The K-Vector approach was discussed in some detail to illustrate the means by which functions may use its advantageous properties. An investigation in to the speed performance was discussed. Results highlighted sorting and building as the primary performance penalties as expected. A way to increase speed performance was investigated with the dynamic k-vector. The implementation showed no noticeable improvement. It was suggested that moving operations to memory operations, rather than disk, may allow more insight in to potential improvements. From purely the number of operations needed, it is expected that there is some improvement from using the dynamic k-vector.

VI.A.3. CANS

Recall the objective of CANS from Section (I):

> An architecture that is conducive to Simultaneous Localization And Mapping (SLAM), path planning, and mission planning.... Such a system involves an image processing methodology to extract relevant and concise information from the environment. This methodology needs to be consistent and robust to varying lighting conditions, types of objects in each scene, and represent sufficiently unique distinctions in the scenes. Additionally, the data management scheme must be optimized for efficient

utilization by SLAM processes, path planning proccesses, and mission planning processes.

It is demonstrated that this architecture works offline (i.e. not in real-time). The system was provided solely attitude accelerations, whereas a real system will likely have accelerometer information as well. This should point to the success in a more difficult problem. In other words, the system restrictions (no accelerometer) here is a much more challenging problem than is likely to be done in practice and we should expect improved performance with this additional information. The path planning and SLAM modules successfully used the data management module operations to use data generated by the image processing module. The mission planning component of the objective was not implemented here. The data used in all the modules are represented by ellipsoids - a primitive shape.

VI.B. Future Work

In this section, future work for the image processing and data management are discussed. In addition, the entire system is discussed. Here, we wish to provide a window into the future as to what is to come.

VI.B.1. Image Processing

The sensor used in this work was an ASUS Xtion Pro. As a cheap RGB-D sensor, it proved to be a very good option. However, the depth data had range limits that generated null data in an indoor environment. Improving the system can first start with better input data. A sensor should be used to provide more consistent depth information for indoor environments. The data generated from the GIS algorithms are more accurate when given uniform images. To improve this information, additional processing should take place in image analysis to create uniform depth maps for these algorithms. Further investigation is needed to detemine the appropriate staturation, noise, and midrange values for the GIS metrics. The image segmentation worked well in the test environment under specified settings. A more robust solution should be investigated. A more robust solution can handle different lighting conditions, locations, distance to observations, and other variations without manual tuning. Some of the predicates used in this work are relatively ad-hoc. Steps should be taken to design predicates that are principle-based to take a more methodical approach to ellipsoid selection.

VI.B.2. Data Management

To better understand the effect of the dynamic K-vector, the database should be stored in RAM. The current implementation stores the database on disk and thus requires reading/writing from/to the disk for any operation interfacing with the database. This will increase speed and potentially shed light on any improvement a dynamic K-vector provides. Recall that m is the 1^{st} polyomial coefficient in the k-vector definition. The dynamic K-vector approach proposed here uses the difference between expected m and the m being used to give a measure of the innacuracy of by-passing the static operations. The effect of simply updating the m with the expected value should be investigated.

VI.B.3. CANS

The demonstration presented is a convincing example of a working end-to-end system, however much work remains. To ensure a more robust system, CANS should be run on a more complicated trajectory. The demonstation presented moved approximately in a 2D plane (though the entire system is 3D). More complicated trajectories may include moving above and through areas of obstruction. In order to test CANS in a real-time environment, the speed performance must be improved. This can be done through the steps mention in Sections (VI.B.1) & (VI.B.2). Additional steps can be to evaluate the redundancy of some operations, hardware specific optimizations, parallelization of operations, and details lying within the SLAM process.

REFERENCES

[1] Kuether, D. J., Morrell, B. J., Chamitoff, G. E., Bishop, M., Mortari, D., Gibbens, P. W., and Coen, M., "Cohesive Autonomous Navigation System," *Guidance Navigation and Control, SciTech 2016 Conference on*, AIAA, 2016.

[2] Durrant-Whyte, H. and Bailey, T., "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms," Tech. rep., Australian Centre for Field Robotics, 2006.

[3] Durrant-Whyte, H. and Bailey, T., "Simultaneous Localisation and Mapping (SLAM): Part II State of the Art," Tech. rep., Australian Centre for Field Robotics, 2006.

[4] Julier, S. J. and Uhlmann, J. K., "New extension of the Kalman filter to non-linear systems," *AeroSense'97*, International Society for Optics and Photonics, 1997, pp. 182–193.

[5] Evensen, G., "The ensemble Kalman filter for combined state and parameter estimation," *Control Systems, IEEE*, Vol. 29, No. 3, 2009, pp. 83–104.

[6] Jaulin, L., "A nonlinear set membership approach for the localization and map building of underwater robots," *IEEE Transactions on Robotics*, Vol. 25, No. 1, 2009, pp. 88–98.

[7] Milford, M. and Wyeth, G., "Persistent Navigation and Mapping using a Biologically Inspired SLAM System," *The International Journal of Robotics Research*, Vol. 29, No. 9, August 2010, pp. 1191–1153.

[8] Nguyen, H. K. and Wongsaisuwan, M., "A study on Unscented SLAM with path planning algorithm integration," *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2014 11th International Conference on*, IEEE, 2014, pp. 1–5.

[9] Richards, A. and How, J., "Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming," *American Control Conference, 2002. Proceedings of the 2002*, Vol. 3, 2002, pp. 1936–1941 vol.3.

[10] Lalish, E., Morgansen, K. A., and Tsukamaki, T., "Decentralized Reactive Collision Avoidance for Multiple Unicycle-Type Vehicles," *American Control Conference, 2008*, IEEE, 2008, pp. 5055–5061.

[11] Chang, D. E., Shadden, S. C., Marsden, J. E., and Olfati-Saber, R., "Collision Avoidance for Multiple Agent Systems," *Proceedings of the 42nd IEEE Conference on Decision and Control*, IEEE, December 2003.

[12] Chamitoff, G. E., Saenz-Otero, A., Katz, J. G., and Ulrich, S., "Admissible Subspace TRajectory Optimizer (ASTRO) for Autonomous Robot Operations on the Space Station," *AIAA Guidance, Navigation, and Control Conference*, AIAA Reston, VA, 2014, pp. 1–17.

[13] Bruce, J. and Veloso, M., "Real-Time Randomized Path Planning for Robot Navigation," *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, Vol. 3, IEEE, 2002, pp. 2383–2388.

[14] Şucan, I. A., Moll, M., and Kavraki, L. E., "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, Vol. 19, No. 4, December 2012, pp. 72–82, http://ompl.kavrakilab.org.

[15] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W., "An Evaluation of the RGB-D SLAM System," *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, 2012, pp. 1691–1696.

[16] Eliazar, A. and Parr, R., "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," *IJCAI*, Vol. 3, 2003, pp. 1135–1142.

[17] Oliver, A., Kang, S., Wünsche, B. C., and MacDonald, B., "Using the Kinect as a Navigation Sensor for Mobile Robotics," *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, ACM, 2012, pp. 509–514.

[18] Meyer, F., "Topographic distance and watershed lines," *Signal processing*, Vol. 38, No. 1, 1994, pp. 113–125.

[19] Otsu, N., "A threshold selection method from gray-level histograms," *Automatica*, Vol. 11, No. 285-296, 1975, pp. 23–27.

[20] MacQueen, J. et al., “Some methods for classification and analysis of multivariate observations,” *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, Oakland, CA, USA., 1967, pp. 281–297.

[21] Felzenszwalb, P. F. and Huttenlocher, D. P., “Efficient graph-based image segmentation,” *International Journal of Computer Vision*, Vol. 59, No. 2, 2004, pp. 167–181.

[22] Cheng, Y., “Mean shift, mode seeking, and clustering,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 17, No. 8, 1995, pp. 790–799.

[23] Houser, C., Bishop, M. P., and Barrineau, P., “Characterizing instability of aeolian environments using analytical reasoning,” *Earth Surface Processes and Landforms*, Vol. 40, No. 5, 2015, pp. 696–705.

[24] Kargel, J. S., Abrams, M. J., Bishop, M. P., Bush, A., Hamilton, G., Jiskoot, H., Kääb, A., Kieffer, H. H., Lee, E. M., Paul, F., et al., “Multispectral imaging contributions to global land ice measurements from space,” *Remote Sensing of Environment*, Vol. 99, No. 1, 2005, pp. 187–219.

[25] Mortari, D., “Search-less algorithm for star pattern recognition,” *J ASTRONAUT SCI*, Vol. 45, No. 2, 1997, pp. 179–194.

[26] OpenNI2, “OpenNI 2 SDK Binaries and Docs,” Online: http://structure.io/openni, November 2015, Last viewed 2015-11-28.

[27] Robertson, A. R., “The CIE 1976 Color-Difference Formulae,” *Color Research & Application*, Vol. 2, No. 1, 1977, pp. 7–11.

[28] Ford, A. and Roberts, A., "Colour space conversions," *Westminster University, London*, Vol. 1998, 1998, pp. 1–31.

[29] Zhang, T. and Suen, C. Y., "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, Vol. 27, No. 3, 1984, pp. 236–239.

[30] Shi, J. and Malik, J., "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 22, No. 8, 2000, pp. 888–905.

[31] Bradski, G., *Dr. Dobb's Journal of Software Tools*, 2000.

[32] MATLAB, *version 8.6.0 (R2015b)*, The MathWorks Inc., Natick, Massachusetts, 2015.

[33] Mortari, D. and Neta, B., "K-vector range searching techniques," 2014.

[34] Wan, E., Van Der Merwe, R., et al., "The unscented Kalman filter for nonlinear estimation," *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, IEEE, 2000, pp. 153–158.

[35] Kraft, E., "A quaternion-based unscented Kalman filter for orientation tracking," *Proceedings of the Sixth International Conference of Information Fusion*, Vol. 1, 2003, pp. 47–54.

APPENDIX A

A.A Image Processing

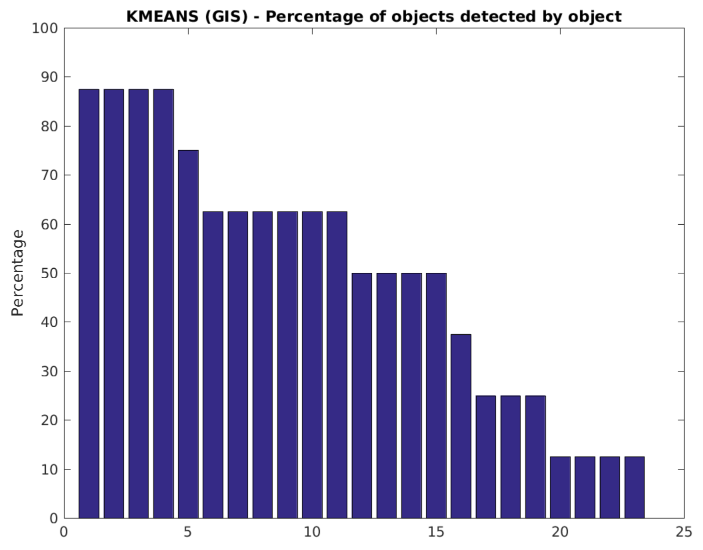


Figure A.1. Repeatability comparison: k-means with GIS

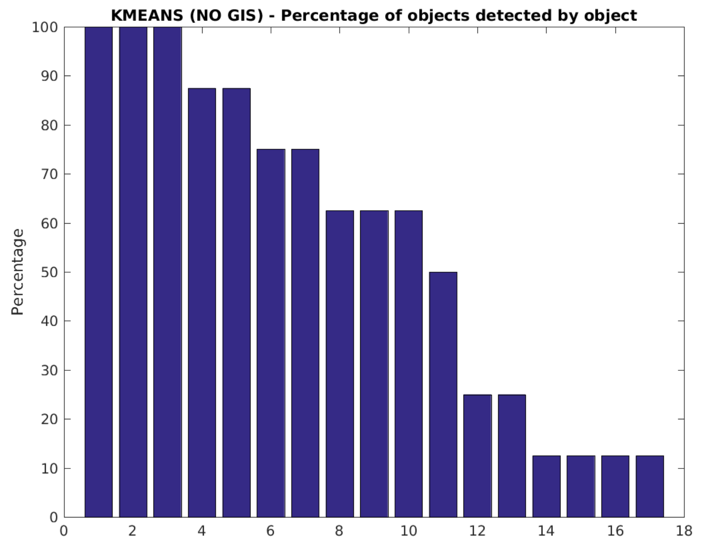


Figure A.2. Repeatability comparison: k-means without GIS

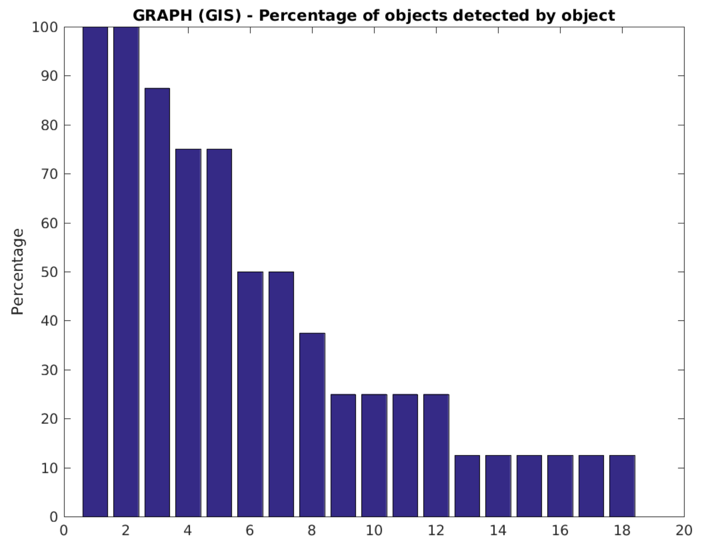


Figure A.3. Repeatability comparison: graph with GIS

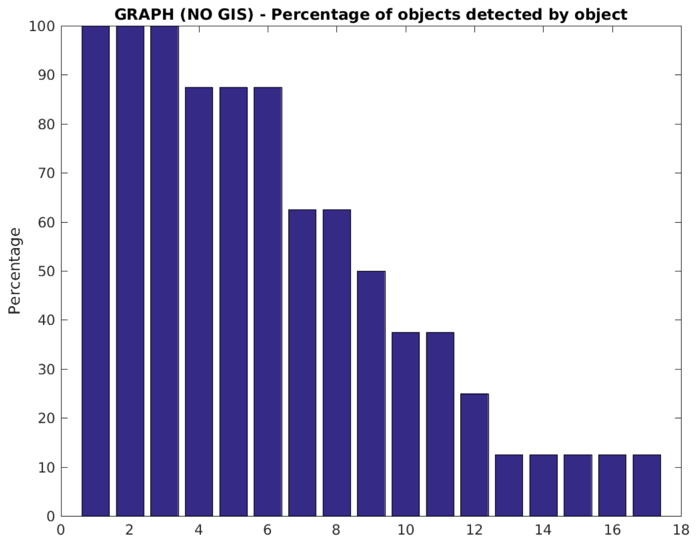


Figure A.4. Repeatability comparison: graph without GIS

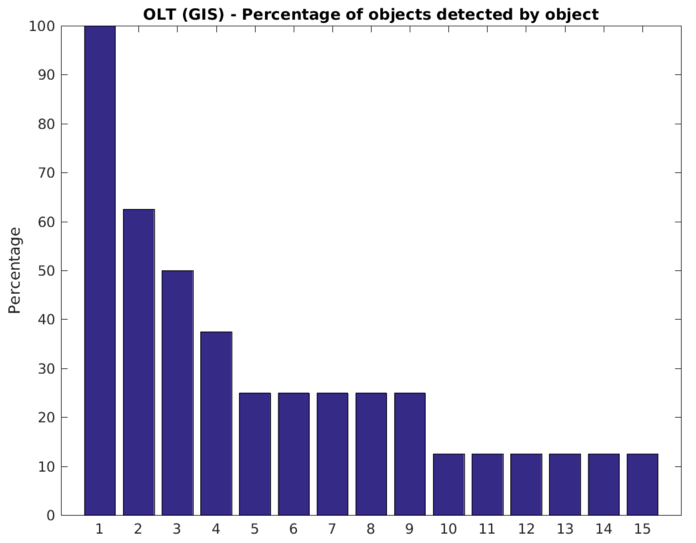


Figure A.5. Repeatability comparison: OLT with GIS

A.B Database Operations' Static Speed Performance

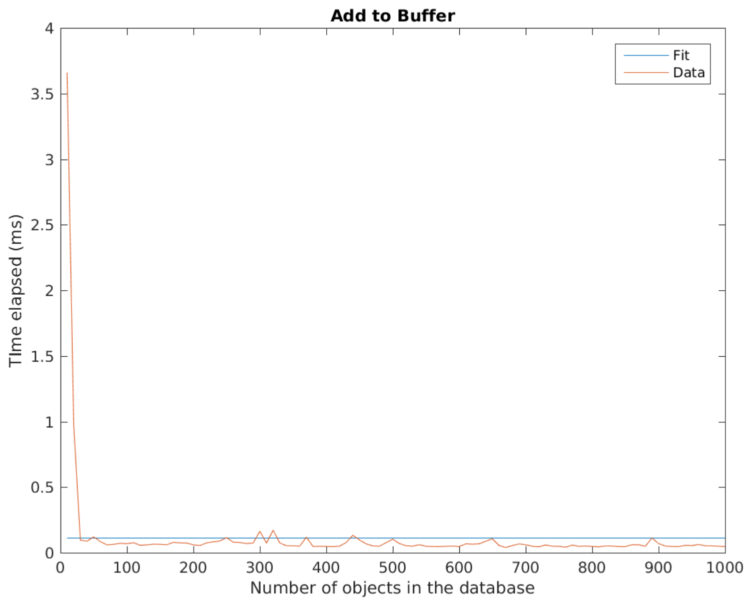


Figure A.6. Add to buffer operational speed

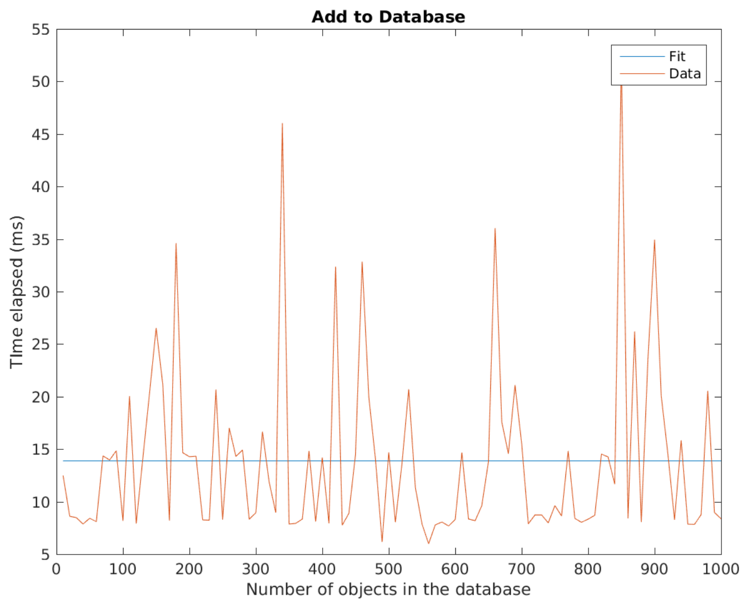


Figure A.7. Add to database operational speed

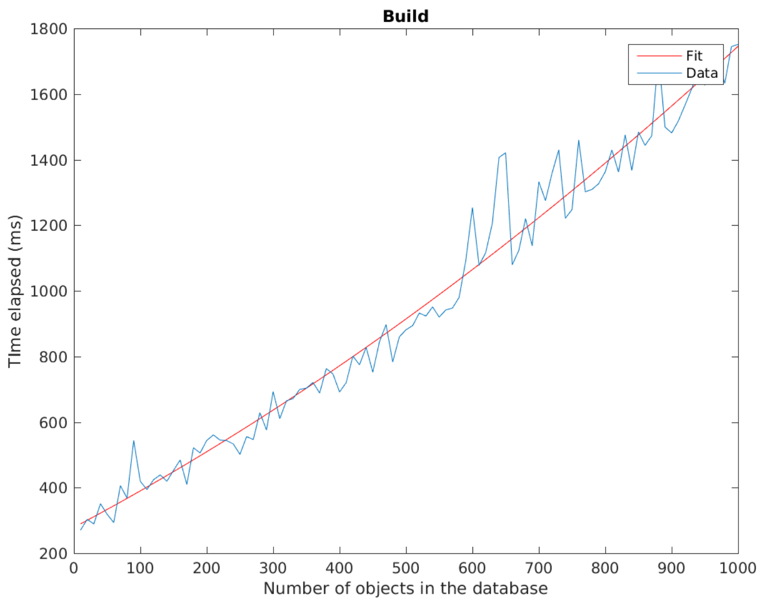


Figure A.8. Build operational speed

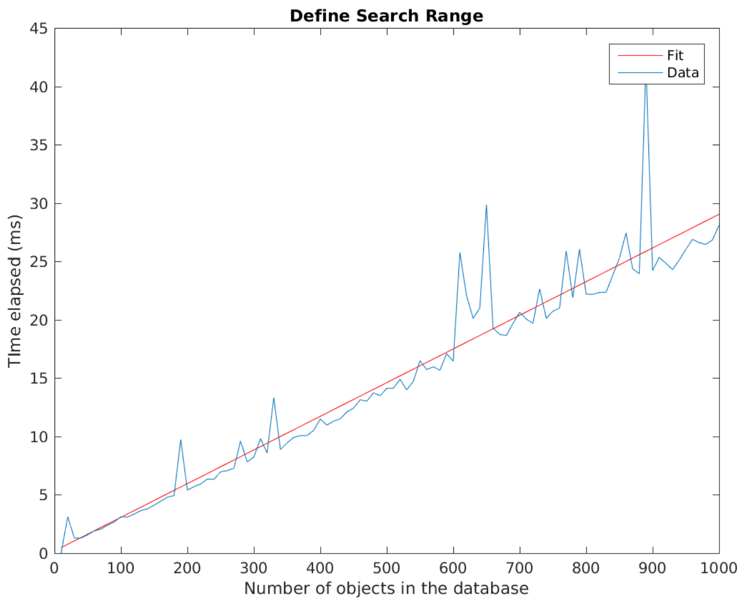


Figure A.9. Define search range operational speed

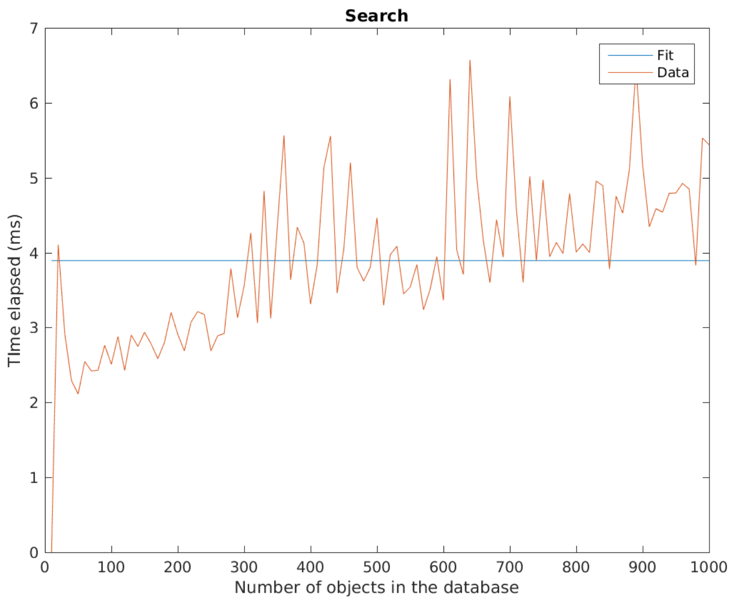


Figure A.10. Search operational speed

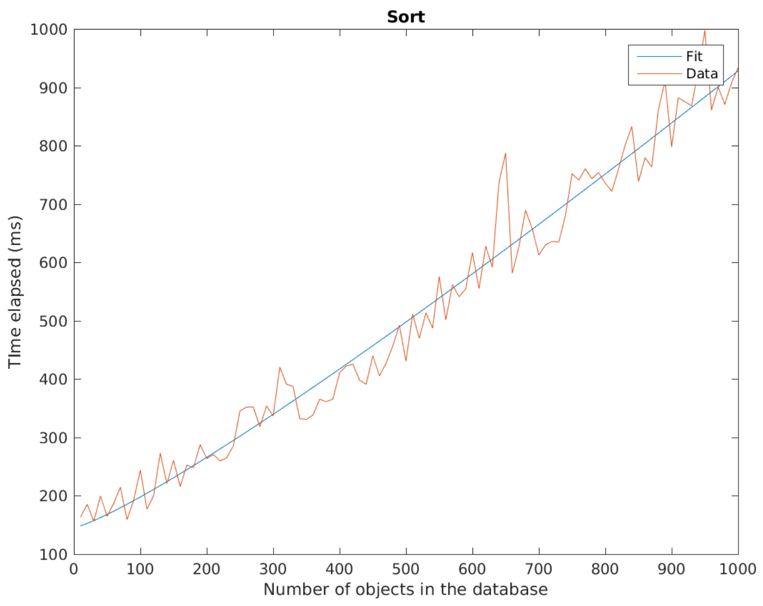


Figure A.11. Sort operational speed

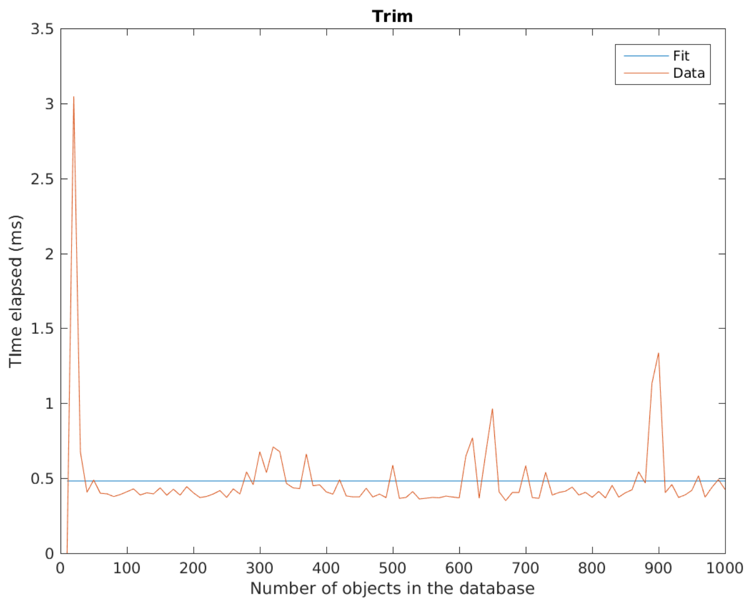


Figure A.12. Trim operational speed

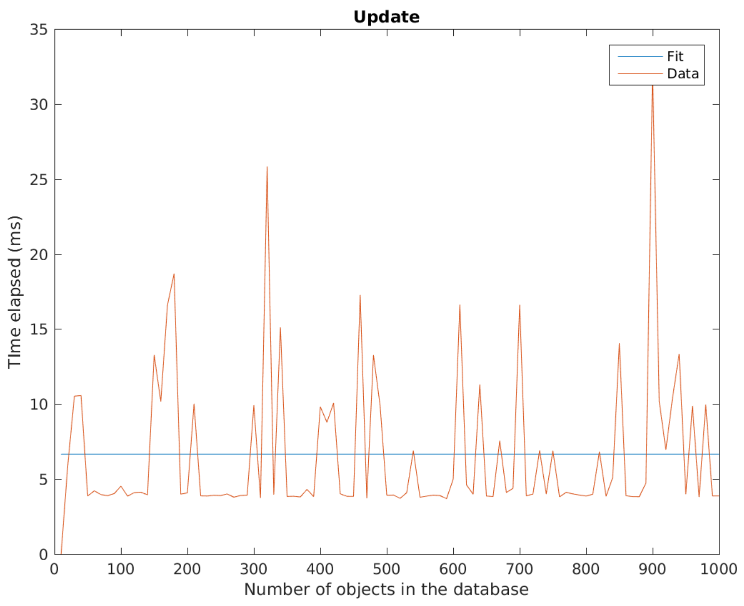


Figure A.13. Update operational speed

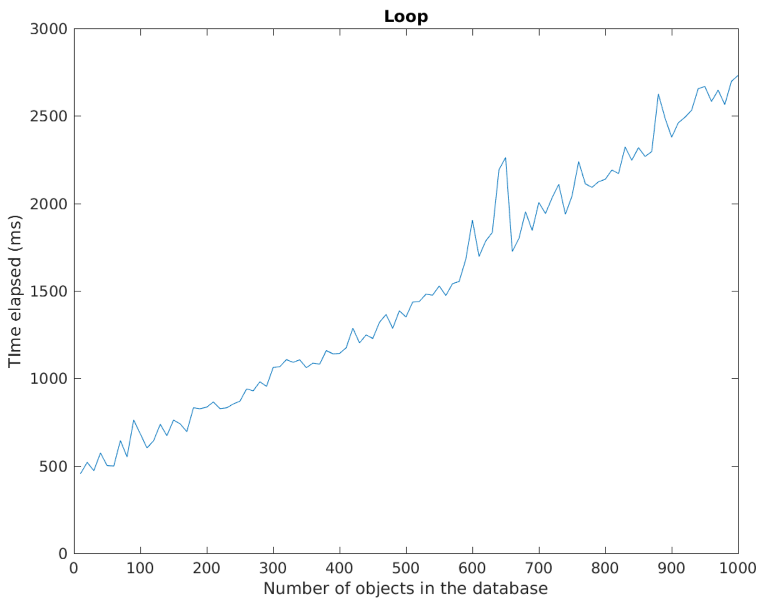


Figure A.14. Loop operational speed

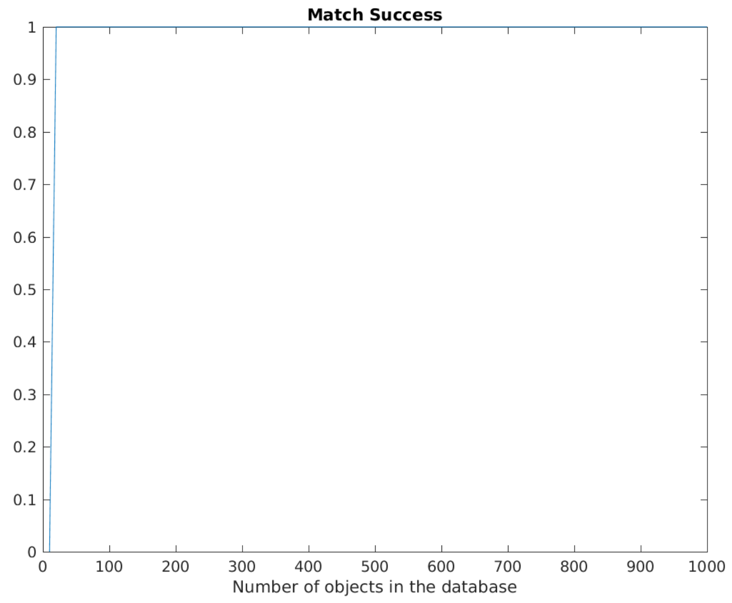


Figure A.15. Match operational speed

A.C Database Operations' Dynamic Speed Performance

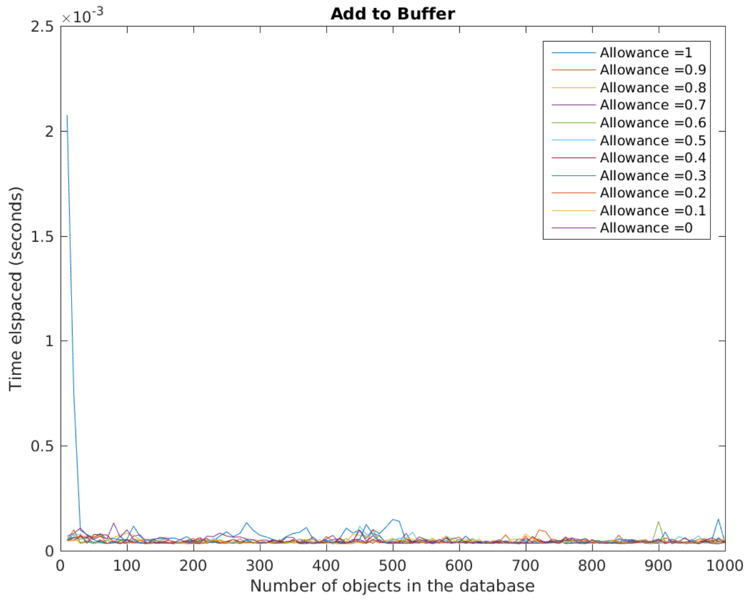


Figure A.16. Dynamic add to buffer operational speed

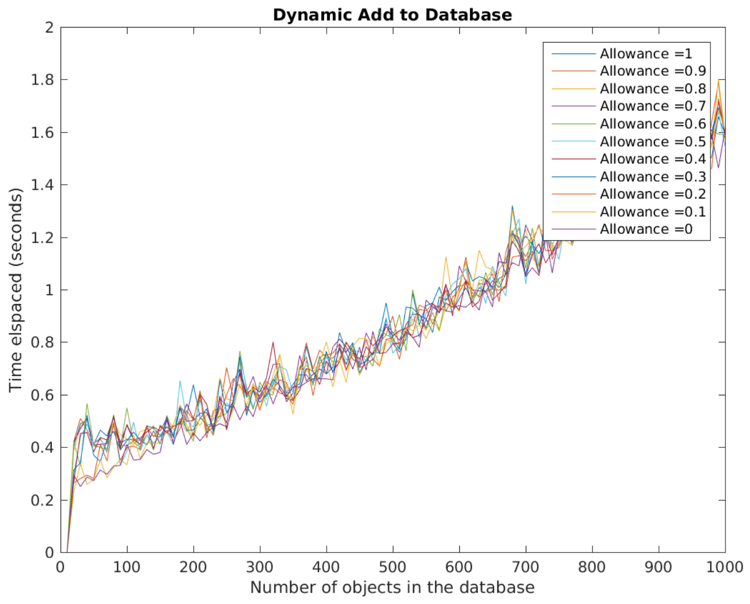


Figure A.17. Dynamic add to database operational speed

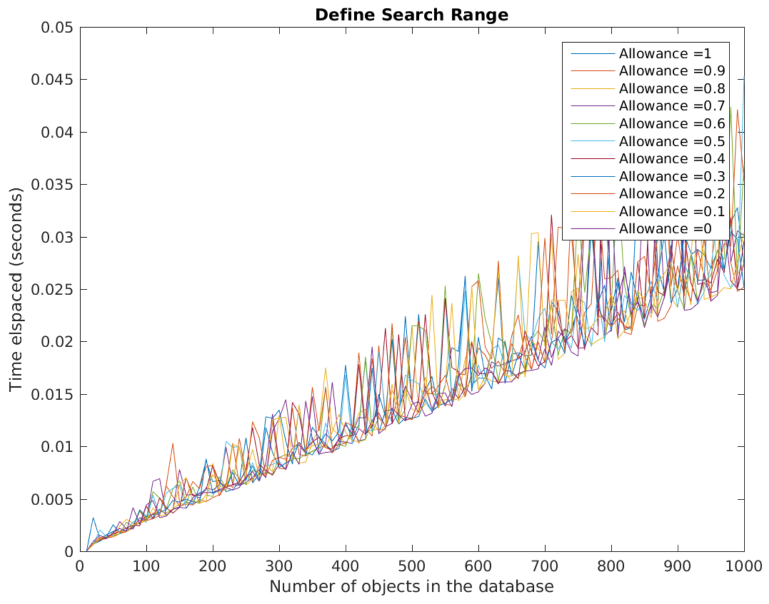


Figure A.18. Dynamic define search range operational speed

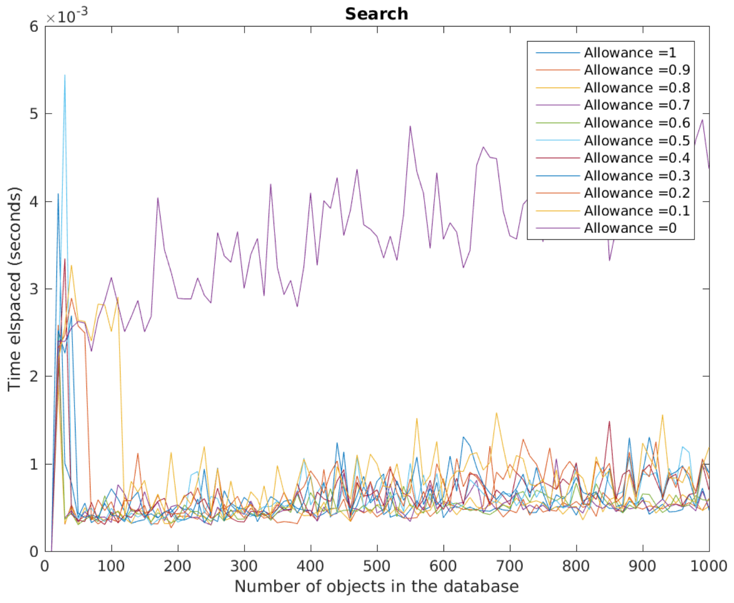


Figure A.19. Dynamic search operational speed

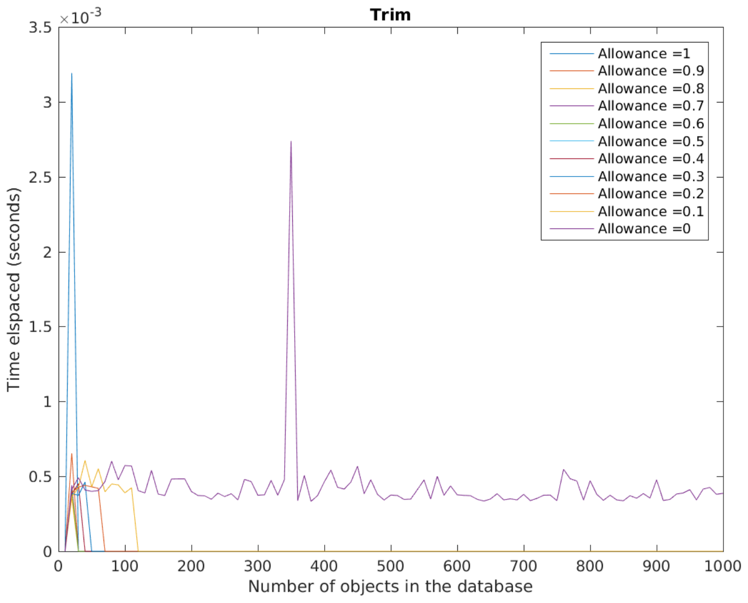


Figure A.20. Dynamic trim operational speed

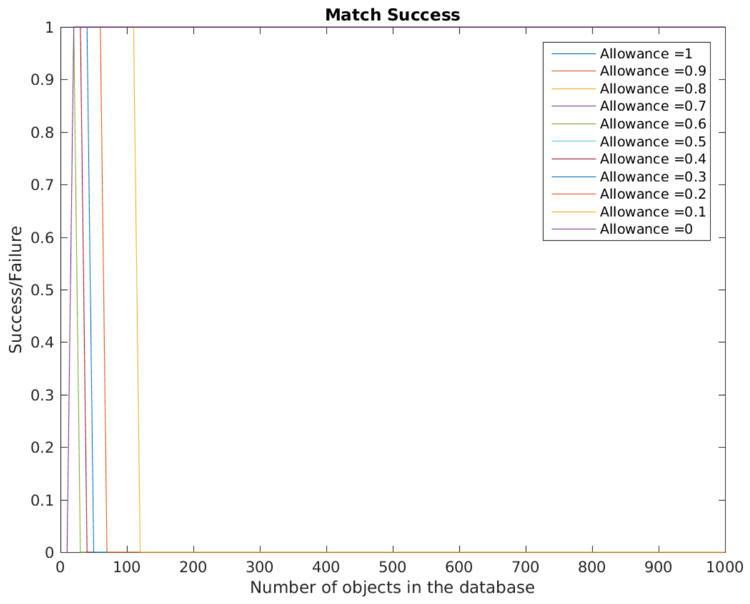


Figure A.21. Dynamic match operational speed

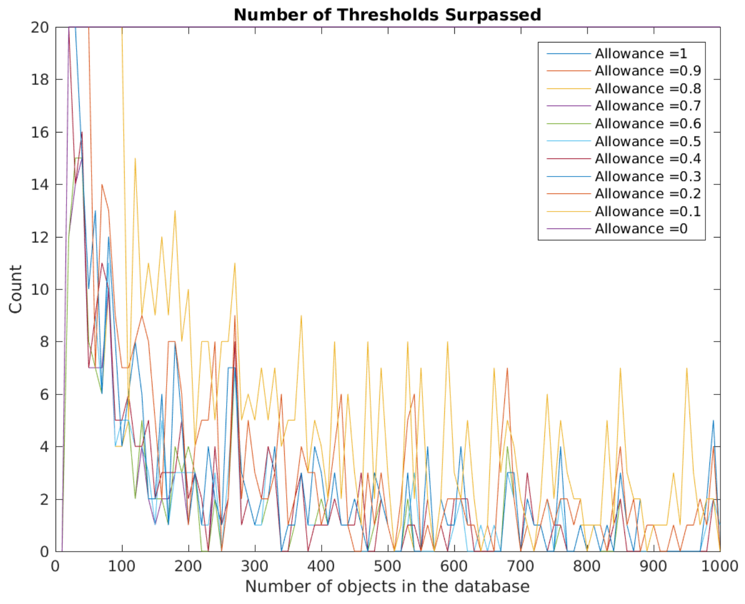


Figure A.22. Dynamic thresholds surpassed

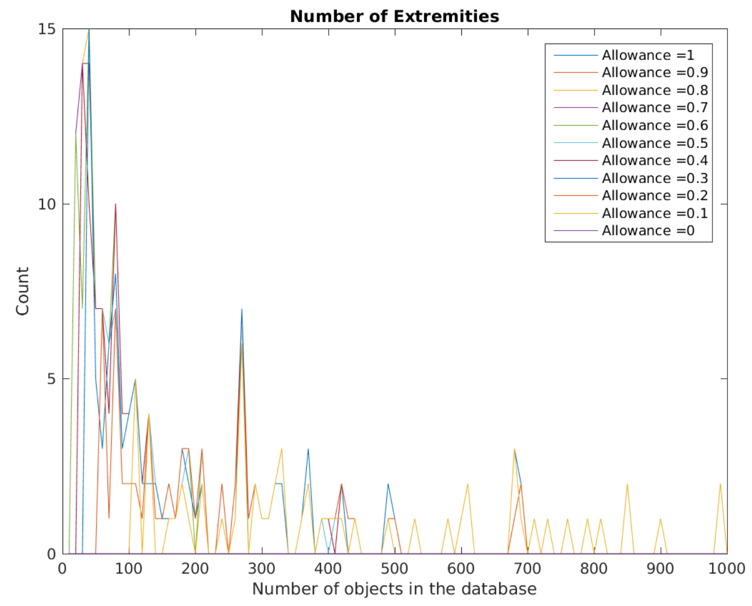


Figure A.23. Dynamic extremeties surpassed

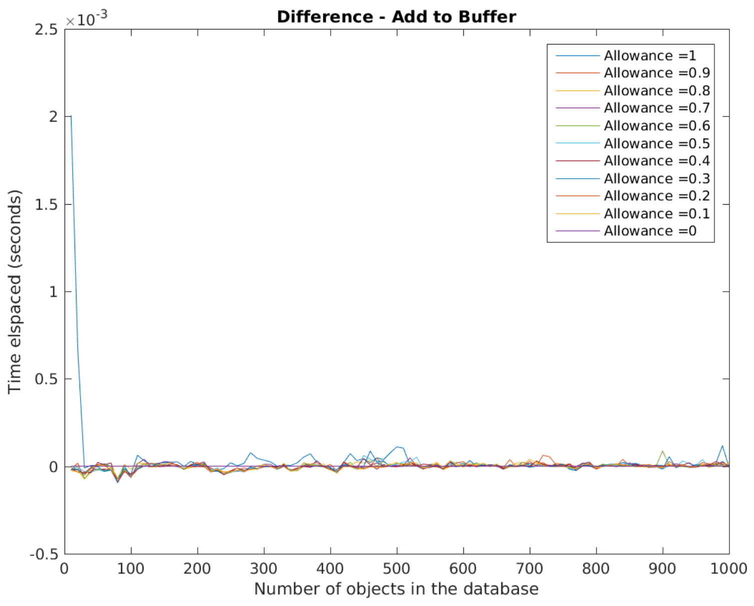


Figure A.24. Dynamic add to buffer operational speed (difference)

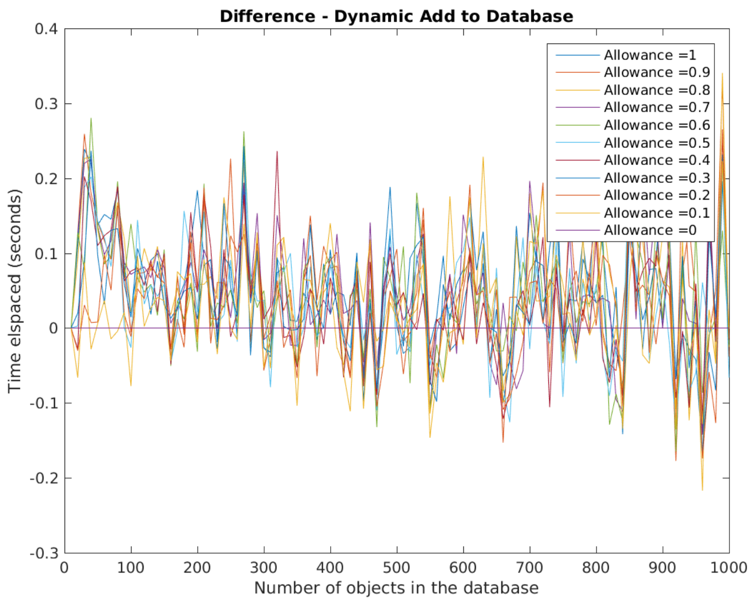


Figure A.25. Dynamic add to database operational speed (difference)

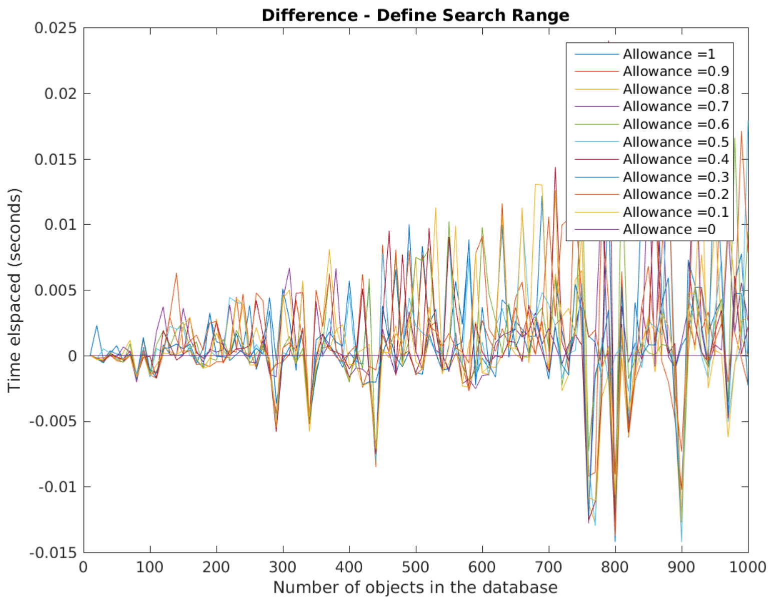


Figure A.26. Dynamic define search range operational speed (difference)

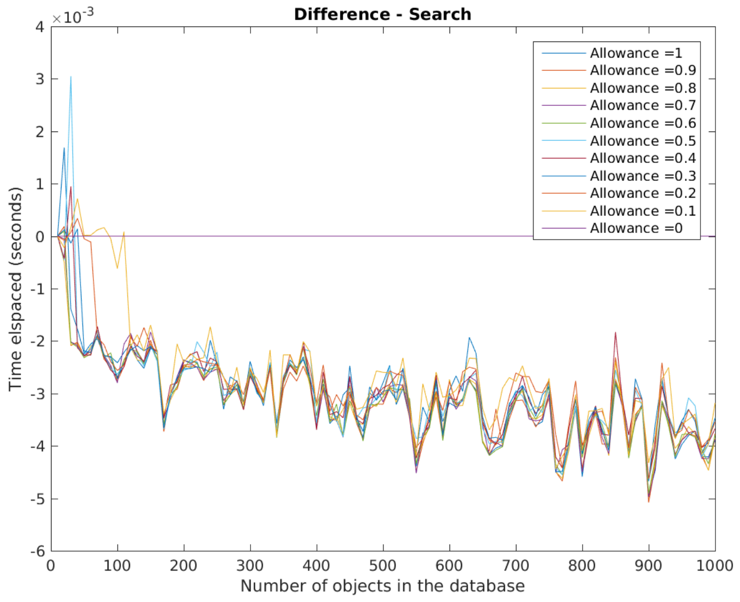


Figure A.27. Dynamic search operational speed (difference)

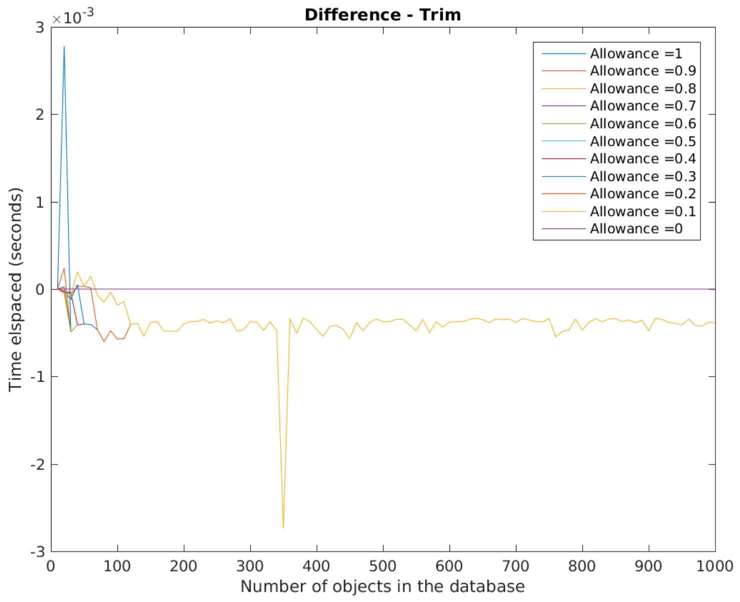


Figure A.28. Dynamic trim operational speed (difference)

A.D CANS Performance

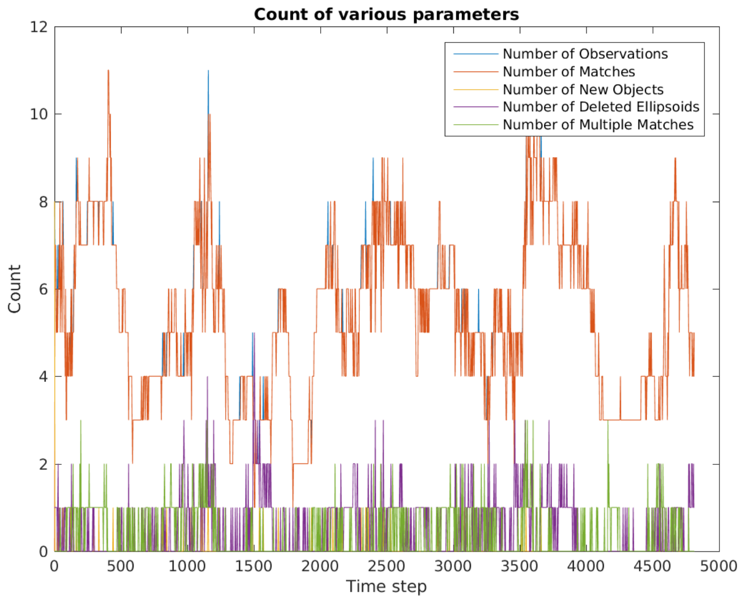


Figure A.29. Count of system parameters

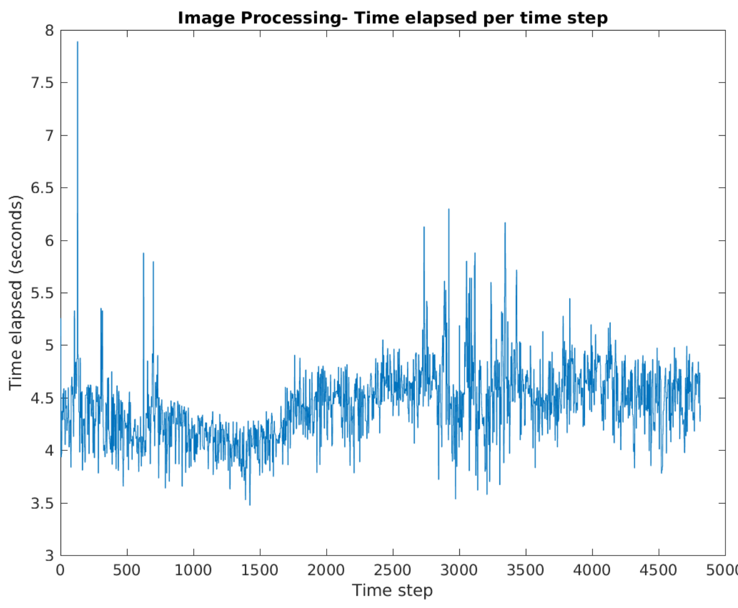


Figure A.30. Time elapsed per time step: image processing

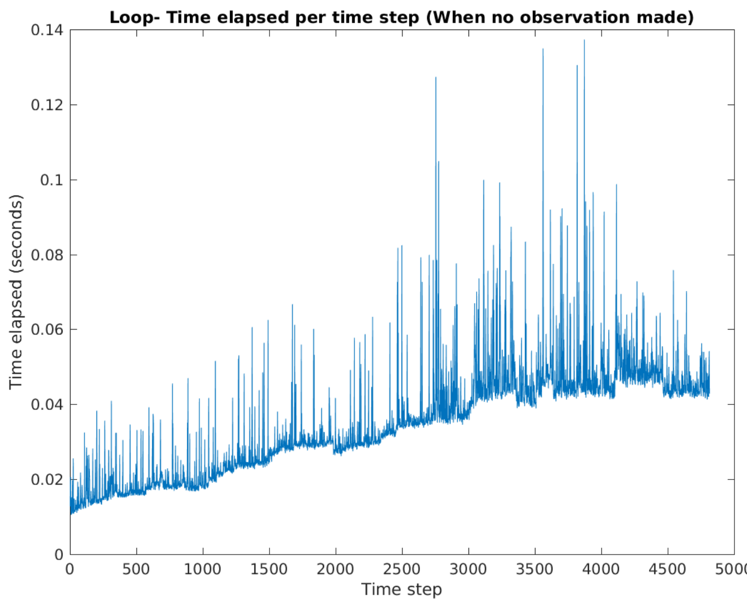


Figure A.31. Time elapsed per time step: entire time step (no observation)

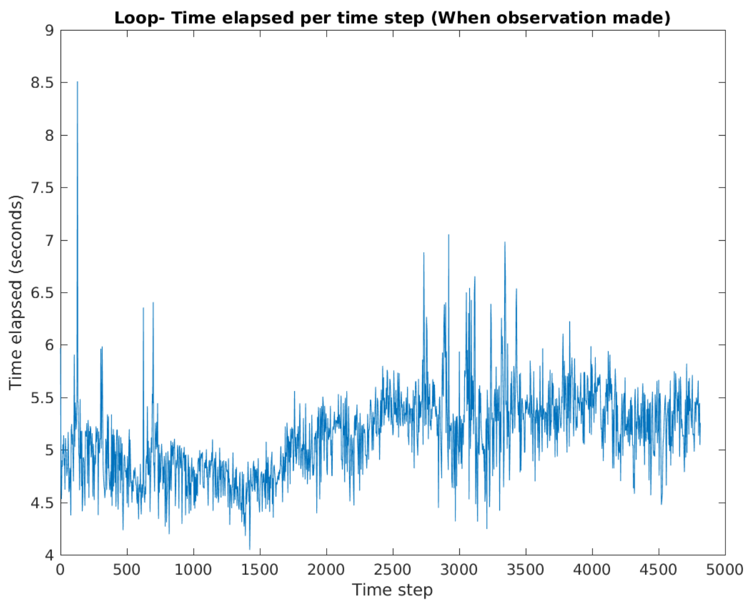


Figure A.32. Time elapsed per time step: entire time step (observation)

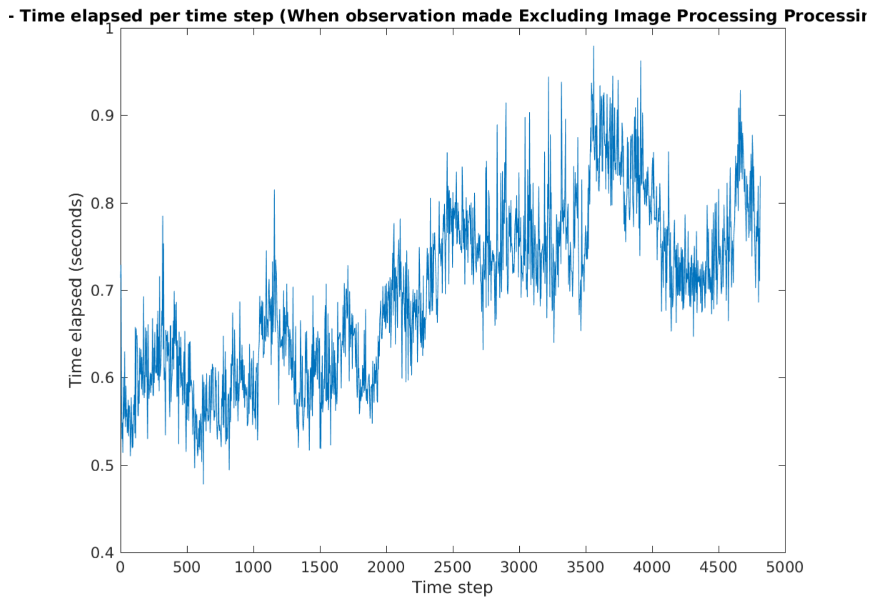


Figure A.33. Time elapsed per time step: entire time step (observation excluding image processing)

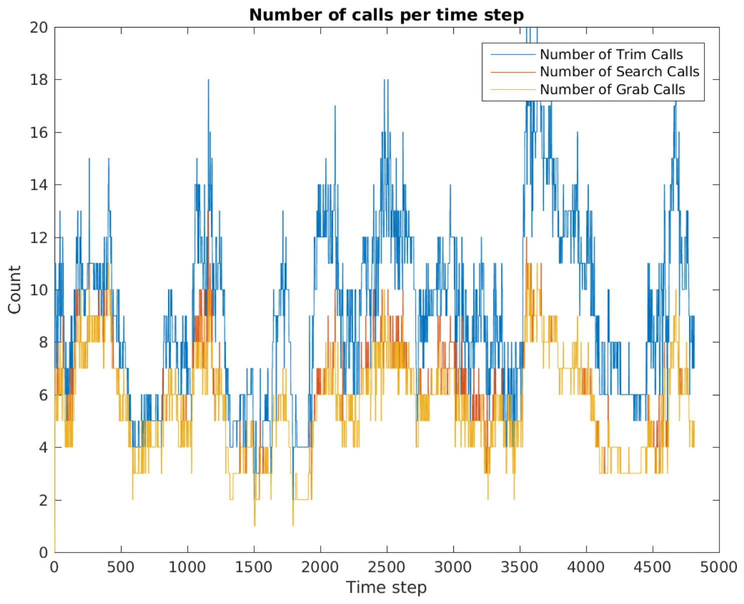


Figure A.34. Number of calls per time step

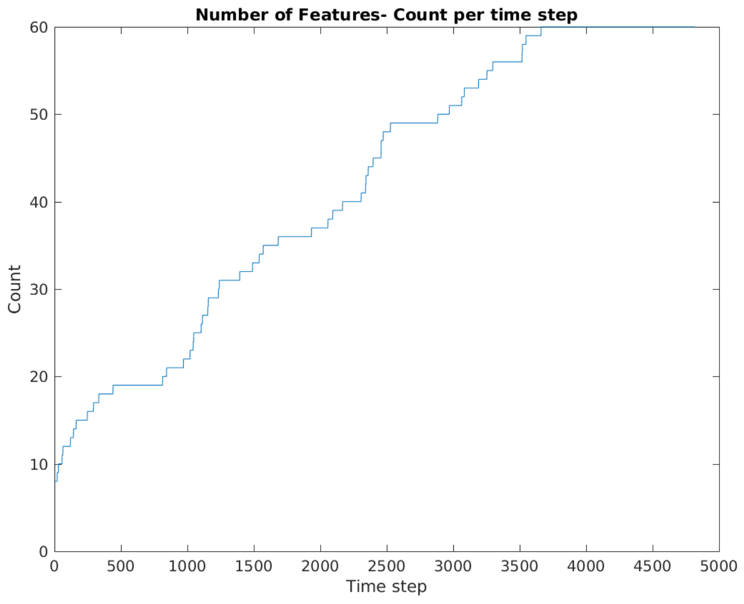


Figure A.35. Number of features in database per time step