# MAPPING, LOCALIZATION, AND NAVIGATION FOR ROBUST AUTONOMOUS INSPECTION OF INDOOR ENVIRONMENTS WITH UNMANNED GROUND VEHICLES

by

Erik Pearson

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Erik Pearson, Candidate

ADVISORY COMMITTEE

Brendan Englot, Chairman Date

Enrique Dunn Date

Hamid Jafarnejad Sani Date

Long Wang Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2023

# MAPPING, LOCALIZATION, AND NAVIGATION FOR ROBUST AUTONOMOUS INSPECTION OF INDOOR ENVIRONMENTS WITH UNMANNED GROUND VEHICLES

## ABSTRACT

The need for autonomous infrastructure inspections performed by mobile robots is becoming increasingly prevalent, to mitigate human error and inspect critical infrastructure with increased frequency, while reducing costs. Accurate mapping, localization and navigation are required to perform autonomous inspection tasks in indoor environments with unmanned ground vehicles (UGVs). Computer vision and precise manipulator control are necessary for successful handheld sensor interactions. This dissertation will discuss completed work towards these capabilities.

Occupancy grid maps are used to condense large volumes of sensor data into manageable and useful data structures for planning and decision-making. Most such maps only consider three possible states for each grid cell: free, occupied and unknown. However, occupancy grid maps' fixed spatial resolution limits their ability to identify sharp changes in occupancy. We propose a novel multi-resolution occupancy mapping algorithm that uses Bayesian generalized kernel inference to distinguish uncertain regions from unknown regions within an occupancy grid, permitting autonomous inspections to address each category separately.

Reliance on electrical power has become a necessity in many parts of the world. Maintaining the equipment used for power distribution is critical. Handheld partial discharge (PD) sensors offer a method for routine inspections to identify degrading hardware before failure. Multiple steps are required to practically enable a mobile manipulation robot platform to perform these inspections.

While simultaneous localization and mapping (SLAM) is effective for navigating an unknown environment, repetitive inspection tasks are often better-served by relying on localization relative to a prior, existing map. By registering current LiDAR data to a global reference map, waypoints assigned to the global frame can be sent to the robot. A framework for localization using prior LiDAR-derived maps will be presented, and leveraged within a novel system architecture supporting multi-session autonomous navigation of a UGV to designated waypoints throughout an indoor environment of interest.

A unique UGV platform has been developed that incorporates a six degree-of-freedom manipulator arm with a custom end-effector to interact with a handheld PD sensor. This integration will be discussed, along with the development and testing of algorithms needed to support its autonomous operations within a substation. Coordination between the UGV and manipulator arm using their respective sensing modalities and computing capabilities is required for successful autonomous inspections. Real-time image processing enables precise localization for robot interactions with points of interest within the environment. Automated manipulator control and PD sensor button-pressing interactions are also discussed. With each of these features implemented, a complete autonomous inspection routine was successfully operated within an electric substation.

Author: Erik Pearson

Advisor: Brendan Englot

Date: 8/16/23

Department: Mechanical Engineering

Degree: Doctor of Philosophy

To my wife who has supported me throughout my pursuit of philosophy.

**Acknowledgments**

Thanks to Alexandre Batard, Benjamin Mirisola, Cameron Murphy, Christine Huang, Connor O'Leary, Franklin Wong, Julia Meyerson, Luisa Bonfim, Matthew Zecca, Noah Spina, Shalemuraju Katari, Shiv Bhatt, Takumasa Dohi, Thomas Colarusso and Thomas Gana for working with me over the years of my research. A special thanks to Shi Bai, Tixiao Shan, Jinkun Wang, John D. Martin, Fanfei Chen, Kevin Doherty, John McConnell, Yewei Huang, Yin Chen, Kyle Hart, Paul Szenher, Xi Lin, Ivana Collado-Gonzalez and Brendan Englot for a wonderful Robust Field Autonomy Lab.

**Funding Acknowledgement:** This thesis was supported in part by the National Science Foundation, grant number IIS-1723996, and by the Consolidated Edison Company of New York, Inc., through the project "Automated Mobile Robot Inspection and Monitoring of Electric Substations," monitored by S. Sagareli, J. Szabo, M. Walther, and D. Gordon.

## Table of Contents

# Chapter 1
# Introduction

## 1.1 Motivation and Problem Statement

Autonomous mobile robot inspections can reduce human errors, provide significantly more data at reduced costs and operate in hazardous environments. Inspections take on many forms depending on the targeted data. Obstacle data in mapping algorithms are inspections of objects within 3D space, while specialized sensors can take recurring measurements to track any changes. Repeatable routines are easier to program into autonomous robotic inspections, therefore this work focuses on rapid robust robotic inspections.

Autonomous mobile robot exploration uses incomplete information about the surrounding environment to evaluate a performance metric that prioritizes the gathering of new data. However, the data gathered can be degraded by factors such as sensor noise and gaps in coverage within the data itself. Recently, predictive occupancy grid mapping algorithms have been developed to address this issue. Such inference-based occupancy grid mapping algorithms can successfully filter noise and fill gaps, while also decreasing the memory required to maintain such data. Simultaneous localization and mapping (SLAM) can be used alongside such mapping tools to curb the localization error that may further exacerbate mapping errors. Given that there is a metric for exploration using mapping by defining new data, there should be a method to perform inspections as well based on data gathered.

While predictive mapping has addressed some of the central issues in occupancy mapping, it has yet to solve a few problematic consequences of applying predictive

inference in this setting. One example is the over-estimation of free space caused by glancing rays, for which the gaps between sparse range observations may be filled with inaccurate predictions of free space. This can lead to incorrect assumptions in the planning and decision-making processes of robot exploration. The boundaries between free and occupied space within occupancy grid maps are also often characterized erroneously. Further inspection could also help this issue, however there are no clear metrics to both begin and end inspection.

Robots have often been programmed for repeatable tasks as each instance just requires the same code. However, repeatable tasks require some consistency between attempts. One such consistency is location within an environment. For unmanned ground vehicles (UGVs) like Clearpath's Jackal, reliably navigating to specific waypoints can open up a whole new world of repeatable tasks. For example, mobile manipulator platforms would be able to perform repeatable manipulation tasks at specified locations around an environment. However, there are currently no simple GPS-denied localization methods used with navigation available publicly.

One such repeatable task is currently being researched by our lab. Consolidated Edison Ltd. have requested an autonomous substation inspection routine for gathering acoustic and transient earth voltage (TEV) measurements on specific panels that are labeled with barcodes. Mobile manipulation has all the necessary hardware for completing this task, however there are many steps towards full implementation.

The goals of this dissertation are to optimize occupancy grid mapping while providing a framework that can be used to separate exploration from inspection as well as build a unique autonomous inspection routine using a UGV with a LiDAR for waypoint navigation with a modified manipulator for inspection tasks.

The main contributions of this work are as follows:

- A novel method to separate the uncertain state from the unknown state while optimizing occupancy grid mapping [1].
- A robust and rapid method to create global reference maps and use them for autonomous 2D waypoint navigation with tolerances for dynamic obstacles [2].
- A hardware and software integration between an unmanned ground vehicle and a robotic manipulator for mobile manipulation in cluttered indoor environments [3]
- A complete autonomous inspection routine using a PD sensor with a custom mobile manipulation platform [4]

# Chapter 2
# Background

## 2.1 Components for Autonomous Inspections

### 2.1.1 Occupancy Maps

Occupancy maps are typically produced from range data. Range data in the most basic form has a sensor origin linked to each endpoint found during the sensing process. One common data structure used to define these endpoints is a point cloud [5]. As the name implies, a point cloud is made up of points within 3D Euclidean space that are located based on a predetermined frame of reference. Due to their simplicity, point clouds are the default data structure for a wide variety of sensors used for mapping. The Robot Operating System (ROS) [6] is frequently used to both capture and visualize point cloud data. Capturing point cloud data can be performed using real world sensors, or it can be simulated using environments such as Gazebo [7].

However, point cloud data presents a few limitations. Firstly, the data can be overwhelming in size. A 32-beam lidar can acquire over one million points per second, which rapidly accumulates when scanning a large environment. Secondly, there is no native free space representation within the point cloud data structure. Further computation is necessary to derive free space from each scan. Thirdly, there is no native representation of obstacle boundaries within a point cloud, and further computation is required to estimate them. Finally, point cloud data is susceptible to corruption by noise and other sources of error.

Considering the limitations of point cloud data, researchers have pursued new data structures to define maps efficiently. A common method in use today is the

occupancy grid map [8], which discretizes 3D space into voxels with a designated spatial resolution. By setting a low resolution we can drastically reduce the size of the data structure, as all points within a given voxel will be tallied together. However, low resolutions reduce our understanding of the environment. Therefore, a balance is necessary between memory efficiency and detail.

OctoMap [9], proposed more than a decade ago, continues to be widely used for memory-efficient 3D occupancy grid mapping. OctoMap is built to take point cloud data as an input, which is processed into a probabilistic map of free and occupied cells at a specified spatial resolution. Successors such as UFOMap [10] or the method described in [11] have been proposed in recent years to optimize both the process of introducing points into the map, as well as the structure of the map.

Occupancy mapping algorithms often permit cells to be represented at multiple spatial resolutions within a single map. One method used to achieve this is called *pruning*, where newly defined cells belonging to the same class are condensed into larger, lower-resolution cells. Pruning has been utilized in several mapping algorithms to reduce the map data while maintaining an accurate representation of the environment [12, 13, 14].

Regardless of the data structure used, sparse data from sensors will, across some resolutions, result in gaps within an occupancy grid map. These gaps are a common issue, as the distance between neighboring sensor rays increases the further an object is from the sensor origin. Therefore, it is often beneficial to implement predictive mapping techniques to infer the contents of an occupancy map between such gaps.

However, these maps require accurate localization to be processed correctly. The errors induced by inaccurate robot localization corrupt the registration of the point cloud associated with a given range scan. This particular source of error has

been studied extensively, with the most common remedy being Simultaneous Localization and Mapping (SLAM) [15].

### 2.1.2 Localization

Mobile robot navigation has been researched extensively to open a pathway for more advanced tasks. However, navigation requires both localization and environmental data to make informed decisions. SLAM has been successful at satisfying both of those requirements. Most SLAM algorithms define a map origin based on the initial position of the robot when it begins its operations. To perform multi-session tasks, we need to ensure a common origin is used for each session.

Autonomous ground vehicles rely on accurate pose estimation for navigation [16, 17]. While there are many methods for performing pose estimation [18], simplifying the task down to a common sensor type reduces the complexity. LiDAR sensors provide enough data for localization with comparative algorithms. Managing thousands of data points from each scan can be daunting from a computational complexity standpoint, however there are methods to reduce the complexity such as semantic labeling. This approach labels and groups a subset of point cloud data together to appear as a single entity, thereby significantly reducing the overall number of comparisons. Using semantics and Random Sample Consensus (RANSAC), [19] performs stable and accurate pose estimation while eliminating dynamic obstacles from comparisons. Semantic modeling can be a powerful tool for pose estimation as proved in [20], which solved global localization by registering a single LiDAR scan overlapped with a camera to a reference map using segmentation and neural network training. Localization can be performed by focusing on one semantic object class while attaining high accuracy in handling the surrounding data [21].

Two forms of multi-session SLAM persist in mobile robotics research. The first

expands the boundaries of a previously defined map [22], while the second revisits the same location from a previous map, where the environment may have changed [23]. Real world environments are not static, which requires maps to be updated from time to time for accurate localization. Therefore, there needs to be some tolerance for robots to use prior maps with outdated information, by either having enough static points of reference on a prior map or updating a prior map during each session. One example of enough reference data is the work produced by Labbe et al. [24] in illumination invariant visual SLAM, where distinctly different visual references are capable of localizing successfully. Another method separates changes between the current map and the prior [25]. Other researchers prefer to update the global reference map such as Zhao et al. [26], who acknowledge active environment changes such as new stores within a mall should be manageable.

Prior maps can be built from many different data sources. One helpful source is 3D models constructed using computer aided design (CAD). Building Information Models can also be used to generate maps for both geometric and semantic localization [27]. Not many real world structures have complete 3D models, however a 3D mesh can be approximately generated from a 2D floorplan for precise robot localization [28]. 3D models of real world environments work well in scenarios where the models already exist. For scenarios that begin with a completely unknown environment, the goal of this work is to provide an easy-to-deploy alternative solution.

### 2.1.3 Navigation

While consistent localization through pose estimation is required for multi-session waypoint navigation, environments are rarely static. One method to handle this is ignoring data unrelated to localization. Indoor artificial landmarks such as fiducial markers can be placed in an environment where other features may change [29].

Visual teach and repeat methods [30, 31] allow for repeated navigation without localization which can function even under seasonal environmental changes [32, 33]. However, robust localization methods can sometimes handle large static changes in the environment.

## 2.2 Hardware

Mobile robots have been used for many inspection, maintenance and repair tasks and come in many forms to optimize the tasks being performed. In just the energy sector alone, hundreds of robots have been designed for such tasks [34]. One special platform that has the potential to address a multitude of tasks using the same hardware is the mobile manipulation robot. Mobile manipulation requires two main hardware components: a base vehicle that can navigate throughout an environment, and a manipulator that can intervene in the environment.

Mobile manipulation has been around for more than 30 years [35] [36] as an autonomous solution for tasks commonly performed by people. However, many tasks have specific hardware requirements to be accomplished, which has resulted in an abundance of mobile manipulation platforms. Larger mobile manipulators such as the Centauro [37], EL-E [38], or Care-O-bot [39] are designed for retrieval and carrying tasks, while smaller options such as Stretch [40], Omnivil [41], and youBot-a [42] often focus more on interaction with a limited environment. The trend focuses on stability, where larger bases can use larger manipulators, while smaller bases are drastically limited in their manipulation capabilities.

Many robotics teams have built custom mobile manipulation platforms, with varied applications in mind. In this dissertation, we describe a platform built to have a small footprint for use within a busy indoor environment, while still being

able to access a large manipulator workspace. For that purpose, Clearpath's Jackal Unmanned Ground Vehicle (UGV) was chosen for the base. Less than two feet in each dimension, this UGV can navigate through busy corridors as easily as a person would. Our Jackal is equipped with a Velodyne VLP-16 "puck" LiDAR, commonly used for mapping and localization. The Kinova Gen3 6-DOF robotic arm was chosen as the manipulator for its 891mm reach, wrist-mounted RGB-D camera, and customizable wrist interface.

The complete mobile manipulation architecture described in this dissertation offers an alternative to prior work by incorporating a long reaching manipulator, multiple sensing modalities, and a base that can navigate through cluttered indoor environments effectively. In the following sections, the hardware and software architectures are described first. Two key performance capabilities are then reviewed for compatibility with the platform, including a confirmation that no degradation of individual subsystems occurs. Finally, recommendations are included for further studies of the platform.

## 2.3 Autonomous Inspection

The complex task of autonomous inspection has been studied extensively for many years such as oil pipeline inspections [43] or construction inspection [44] to more modern tasks such as underwater structure inspection [45] and damage type visual inspections [46]. As summers become hotter and our reliance on electricity increases, frequent inspections will become increasingly necessary to reduce outages. Therefore, electric substation inspection robots are being developed [47] to safely manage the anticipated rise in demand.

Methods vary for robotic substation inspection. Computer vision has been

utilized on mobile robots to read analog sensor measurements remotely [48]. Rail mounted inspection robots can correct their pose adaptively to ensure successful inspections [49]. AI is also used to optimize autonomous mobile inspection robots [50]. Collaborative data analysis offers a new perspective for autonomous substation inspections [51]. Outdoor substation inspections require weather independence, including in snowy conditions [52]. The characteristics that connect these substation inspection robots are their ability to navigate through the substation environment and to collect meaningful data.

The most meaningful data in an inspection varies across different environments. Past research on electric distribution substation inspections has pursued methods for identifying potential failures before they occur, using transient earth voltage (TEV) and acoustic measurements [53] or creating an entire condition monitoring methodology [54, 55]. This dissertation offers a method to gather acoustic and TEV measurements from switchgear panels autonomously using a mobile manipulation platform in a low-clearance indoor environment. Specifically, the work that follows focuses on inspecting pothead compartments as a representative test case.

Localization is important for autonomous inspection. While the base vehicle of our mobile manipulation platform can localize using lidar within a global reference frame, making physical contact with precise locations, via visual localization, is also necessary to perform successful inspections. There are many methods to localize objects of interest using computer vision [56], requiring both detection and classification.

While there are many custom solutions for localization using computer vision, some inspection tasks require using the least intrusive methods possible. The partial discharge (PD) sensor used in this work scans a barcode on a square sticker which can be placed in a reasonable location within the environment. Rather than including a

separate localization method, barcode detection was considered ideal. Many options are available for barcode identification, such as the python package OpenCV. The underlying algorithm used by OpenCV is based on directional coherence and looks for patches of high gradient orientation coherence with similar gradient directions. After some filtering, the final result is bounded by a minimum rectangular area to output the corners of the barcode. However, this would only work in ideal conditions, with limited glare and minimal curvature of the barcode. Other methods have been proposed to handle more difficult scenarios [57].

Robotic manipulation has varied over the years as the hardware has changed significantly, resulting in different planning and control methods [58]. Early work offered methods to plan trajectories based on simple one degree-of-freedom robots [59]. Higher complexity from more degrees of freedom introduced sampling based motion planners, such as those using rapidly exploring random trees (RRT) [60, 61, 62]. Robotic manipulation can be performed as either absolute positioning using joint angles or relative positioning typically in Cartesian space. Absolute positioning can be as simple as understanding the current position through encoders and simply moving each joint to the desired final angle. Relative motion, however, requires more complex control. Control methods have been built for Cartesian motion planning using neural networks [63], optimization [64] and model predictive control [65]. Extensions to these motion planners include obstacle avoidance [66] and safe human-robot interactions [67]. This dissertation will focus on a few effective methods for Cartesian motion without requiring these extensions.

# Chapter 3

# Improving Obstacle Boundary Representations in Predictive Occupancy Mapping

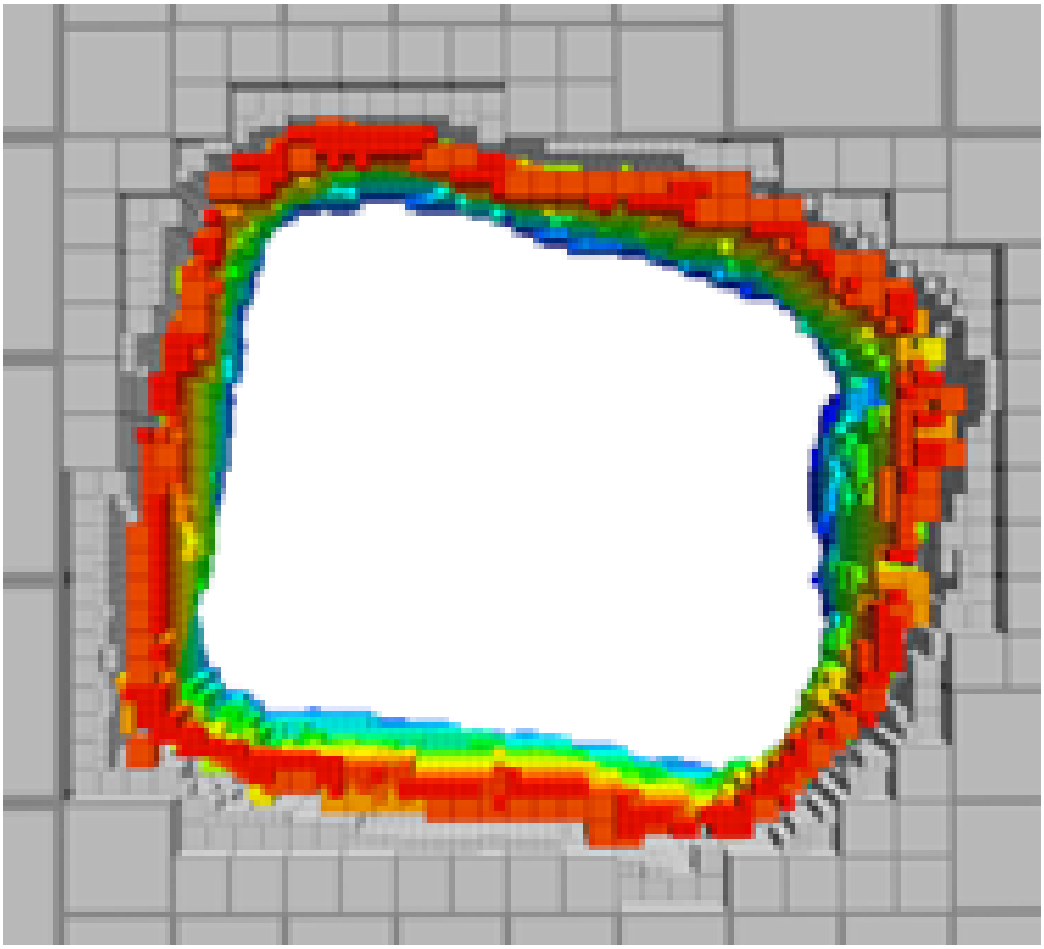


Figure 3.1: Example of the proposed algorithm mapping a column in an underground mine, with free space outside the column shown in gray.

## 3.1 Assumptions

In this section, we will focus on errors that derive from additive noise in the sensing process itself, which is frequently assumed to be Gaussian [68], but can take on other

characteristics. We will assume our robot's state is fully observable.

### 3.1.1 Learning-Aided Occupancy Mapping

There are many methods for predicting the missing contents of 3D occupancy grid maps, however the majority of them frame occupancy mapping as a supervised learning problem. Gaussian processes have been widely used for this purpose, and they can accommodate a variety of kernel functions; of particular note is their compatibility with a sparse covariance function intended for exact inference over large datasets [69]. In most learning-aided mapping approaches, predicting the occupancy probability of cells not directly observed by sensor rays is influenced by the proximity of existing data to the query point. Each scan from a sensor can be interpreted as a set of "hit points" that define occupied regions and sensor rays that define free space.

Probabilistic inference can also address noise and other sources of uncertainty in occupancy mapping. Warped Gaussian processes [70] and Rao-Blackwellized particle filters [71] can account for the localization uncertainty captured by SLAM, while accurately representing occupancy probabilities across a grid map. Occupancy grid mapping has also been made more compatible with the map corrections required by SLAM in [72], where previous sensor observations are moved within an occupancy map as SLAM corrections are made. Due to the added expense of keeping track of all sensor observations so they can be moved if necessary, the majority of inference-based occupancy mapping algorithms, including ours, assume a robot's localization is accurate, but that its range sensor is noisy.

Many occupancy mapping algorithms use Gaussian Processes (GPs) to infer the missing contents of gaps caused by sensor inadequacies [13, 73, 74, 75, 76]. Due to the computational complexity of GPs, Bayesian generalized kernel (BGK) [77] inference has been proposed as an efficient alternative technique for applying Bayesian

nonparametric inference to occupancy grid maps [12, 14]. Other inference methods that address the poor scalability of GPs include integral kernels [78], Markov random field ray propagation [79], Hilbert maps [80, 81], Hilbert maps with deep learning [82], decomposable radial kernels [83], confidence rich grid mapping [84], and a method that employs spherical cells rather than voxels [85].

Another recently proposed approach uses the agreement/disagreement of nearby sensor observations to respectively stretch or compress the kernel [86]. The adaptive kernel inference mapping algorithm (AKIMap) assumes the gaps between sensor observations to be a greater source of inaccuracy than sensor noise. Accordingly, the AKIMap algorithm achieves gap-filling by expanding the kernel's influence toward similar occupancy states and compressing the kernel's influence away from dissimilar occupancy states.

### 3.1.2 Free Space Representation

One aspect of occupancy grid mapping that distinguishes it from other classification problems is the typical use of three classes to categorize the contents of map cells: free, occupied, and unknown. To frame occupancy mapping as a binary classification problem amenable to solution by supervised learning algorithms, several learning-aided occupancy mapping algorithms focus specifically on classifying *occupied* cells, with the opposite class simply being "not occupied". Such algorithms typically do not quantify the accuracy with which free space can be predicted, focusing instead on the accuracy with which obstacles are mapped [73, 74, 75, 79].

Predictive occupancy mapping algorithms that classify free space adopt a variety of approaches to interpreting a robot's range sensor observations. The most common method is to define an evenly spaced distribution of points between the sensor origin of a range scan and the "hit point" at the end of each sensor ray, which

are interpreted to be sensor observations of free space [13, 14, 76, 78, 84]. AtomMap [85] also adopts this approach, but along the portion of a ray close to the hit point, it computes the normal to the nearby obstacle surface and produces two more free cells normal to the surface. Another method to define observations of free space is to randomly sample points between the sensor origin and the hit point of a ray. This approach requires certain attributes within the algorithm, such as kernel stretching to account for sparsity among the data, but was utilized by [80, 81, 86]. Guo et al. [83] used a Poisson distribution along each ray.

A final method for defining observations of free space is to utilize the sensor rays themselves, rather than representing free space observations using points. Both GP Occupancy Mapping (GPOM) [73] and BGKOctoMap-L [12] do this successfully by using the point-to-line distance, i.e., the shortest distance between a point of interest and a given sensor ray, to define a single free space observation contributed by that ray in the evaluation of a given query point. By adopting this approach, BGKOctoMap-L is able to mitigate the over-sampling of free space that would otherwise result from high-resolution interpolation along each sensor ray.

### 3.1.3 Unknown & Uncertain Representation

The third classification commonly used in occupancy grid mapping, *unknown*, typically applies to cells that have not yet been observed by a robot's sensor. In this paper, we will argue that predictive occupancy mapping benefits from two separate types of unknown classification: the standard definition of unknown in which insufficient data has been procured about a given region, and *uncertain*, where conflicting data makes accurate classification of map cells challenging. Although it has not been labeled as the uncertain class previously in predictive inference-based occupancy mapping, several previous algorithms have captured what they define as the unknown class, ap-

plying it when there is conflicting data. However, a few algorithms created methods to define unknown as missing data.

By explicitly defining free space, [11] was able to infer unknown cells as those remaining in the map apart from the cells classified as occupied and free. This "leftover" method is widely used in occupancy grid mapping, by which all map cells are defined to be unknown at the start of a mapping exercise, until sensor rays pass through them or near them, and they are gradually eliminated and replaced by free and occupied cells.

Another method to define unknown in occupancy mapping was proposed by Jadidi et al. [87], in which two separate maps are generated and then fused together. The first map estimates occupied and not occupied classes, while the second map estimates free and not free classes. When the two maps are merged together, the resulting map contains free and occupied cells contributed by each component map, and the remaining cells are categorized as unknown. This merging method originally encountered problems with overestimation or underestimation of a given class due to the selected kernel functions being applied to separate inference problems in isolation. A revised method was later proposed to solve this issue in [88], by merging the two component maps into a unique continuous occupancy map. Although most mapping algorithms ignore under- and over-estimation of the free and occupied classes, we aim to make an improvement specifically to address that issue. Additionally, in the sections that follow, we will describe a methodology to suitably define both unknown and uncertain classes separately within a predictive occupancy map, to achieve descriptive and accurate inference.

## 3.2 Bayesian Generalized Kernel Inference

### 3.2.1 Counting Sensor Model

Our proposed algorithm is structured similarly to the BGKOctoMap-L framework proposed by a subset of the authors, in [12]. Therefore, we make similar assumptions about our workspace, where measurements are defined as $\mathcal{Y} = \{y_1, \ldots, y_N \mid y_i \in \{0, 1\}\}$ and the values are assigned based on whether the measurements are a ray or hit point, which represent free space and occupied space respectively. Each $y_i \in \mathcal{Y}$ has a corresponding location denoted as $\mathcal{X} = \{x_1, \ldots, x_N \mid x_i \in \mathbb{R}^3\}$, such that all $N$ range measurements can be described fully in the form of the tuple $(x_i, y_i)$. Map cells are also assumed to be indexed by $j \in \mathbb{Z}^+$.

However, our proposed algorithm, BGKOctoMap-LV, differs from the the prior work in the way we define the occupancy of a map cell. In [12], each cell was given a probability of occupancy defined by $\theta_j$ with Bernoulli likelihood

$$p(y_i \mid \theta_j) = \theta_j^{y_i}(1 - \theta_j)^{1-y_i}, \tag{3.1}$$

where the posterior function $p(\theta_j \mid \mathcal{X}, \mathcal{Y})$ was used to estimate occupancy. This function was obtained using a conjugate prior, where $\theta_j$ was assumed to be a Beta distribution, resulting in the following occupancy estimator and variance:

$$\mathbb{E}[\theta_j] = \frac{\alpha_j}{\alpha_j + \beta_j} \tag{3.2}$$

$$\mathbb{V}[\theta_j] = \frac{\alpha_j \, \beta_j}{(\alpha_j + \beta_j)^2(\alpha_j + \beta_j + 1)} \tag{3.3}$$

The hyperparameters $\alpha_j$ and $\beta_j$ were defined to simplify the joint measure-

ment likelihood of each cell. Observations are only added to the "counts" of these parameters if they fall within the region $R_j$ that influences cell $j$.

$$\alpha_j := \alpha_0 + \sum_{i,\, x_i \in R_j} y_i \tag{3.4}$$

$$\beta_j := \beta_0 + \sum_{i,\, x_i \in R_j} (1 - y_i). \tag{3.5}$$

By this definition, $\alpha_j$ represents the total count of hit points within region $R_j$, while $\beta_j$ is the total number of occurrences of rays passing through the same region $R_j$. Each term also has a prior defined by $\alpha_0$ and $\beta_0$.

In that previously used estimation procedure, $\theta_j$ is assumed to have a Beta distribution. However, for simplification, we instead assume a Bernoulli distribution. We also define a new parameter for each cell $j$ to be the occupancy state $m_j \in \{0, 1\}$, which follows from the definition of measurements $y_i$, where 0 represents the free or unoccupied state while 1 is defined as occupied. This distribution's probability mass function can leverage the same $\alpha_j$ and $\beta_j$ parameters described previously.

$$P(m_j) = \begin{cases} \dfrac{\alpha_j}{\alpha_j + \beta_j}, & \text{if } m_j = 1 \\ \dfrac{\beta_j}{\alpha_j + \beta_j}, & \text{if } m_j = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

With this new distribution we can estimate occupancy and the variance of occupancy. We will use the term $\mu_j$ to describe the occupancy estimate in the equations below.

$$\mathbb{E}[m_j] = \sum_{m_j=\{0,1\}} m_j \; p(m_j) = \frac{\alpha_j}{\alpha_j + \beta_j} = \mu_j \tag{3.7}$$

$$\mathbb{V}[m_j] = \sum_{m_j=\{0,1\}} (m_j - \mu_j)^2 p(m_j) = \frac{\alpha_j \; \beta_j}{(\alpha_j + \beta_j)^2} \tag{3.8}$$

The expected value of $m_j$ is equal to that of the Beta distribution $\theta_j$, which suggests a reasonable assumption when using the new Bernoulli distribution. However, while the variance estimates are similar, they vary enough to create unique perspectives. The variance of a Beta distribution will tend to zero as more data is captured, even when $\alpha_j$ and $\beta_j$ are equal. The variance of a Bernoulli distribution, however, will lie at a maximum when $\alpha_j = \beta_j$, regardless of how much data has been gathered, serving our goal of using map cell variance to characterize uncertain cells with conflicting observations.

### 3.2.2 Free Space using Rays

Using continuous rays permits an accurate representation of free space, and avoids the potential pitfalls of interpolating along a ray to generate evenly spaced free points from the sensor origin to a hit point. If the free points are discretized too finely, multiple free points will land within individual map cells, outweighing the influence of hit points. If they are discretized too coarsely, cells along a ray may have no points introduced into them, even though a sensor ray has passed through. These problems can be avoided by using the ray itself to represent free space, rather than interpolated points.

Our prior work on BGKOctoMap-L [12] used a continuous ray from the sensor origin to the hit point to define free space, where each map cell only sees the influence of the nearest point on each range beam. The nearest point was found by searching

along the length of the ray using the following equations:

$$x_{free} = \begin{cases} P, & \text{if } \delta < 0 \\ P + \delta \dfrac{\overrightarrow{PQ}}{|PQ|}, & \text{if } 0 < \delta < 1 \\ Q, & \text{if } \delta > 1 \end{cases} \tag{3.9}$$

$$\delta = \frac{\overrightarrow{PQ} \cdot \overrightarrow{Px_*}}{|PQ|}, \tag{3.10}$$

where $P$ is the sensor origin, $Q$ is the hit point, and $x_*$ is the map cell center. When a map cell lies behind the origin of a ray, the nearest point will be $P$, whereas if a map cell lies beyond the end of the ray, the nearest point will be $Q$. However, if the map cell occurs somewhere between $P$ and $Q$ within 3D space, then the nearest point will be determined using $\delta$. These assumptions are also adopted in BGKOctoMap-LV.

### 3.2.3 Kernel Estimation

While the discretization provided by occupancy grid maps is useful in many respects, it can also reduce the accuracy and precision captured by the underlying sensor data. Therefore, we assume there is a smooth distribution of sensor data over any given map cell which can be considered a region of influence around each cell. BGKOctoMap-L uses the following sparse kernel to assume a distribution from a cell center [69]:

$$k(x, x') = \begin{cases} \sigma \left[ \frac{2+\cos(2\pi \frac{d}{l})}{3}(1 - \frac{d}{l}) + \frac{1}{2\pi}\sin(2\pi \frac{d}{l}) \right] & \text{if } d < l \\ 0 & \text{if } d > l \end{cases}, \tag{3.11}$$

where $d = \parallel x - x' \parallel$ is the distance between a map cell's center $x$ and a nearby sensor observation $x'$. A constant hyperparameter $\sigma$ is used to define the strength of

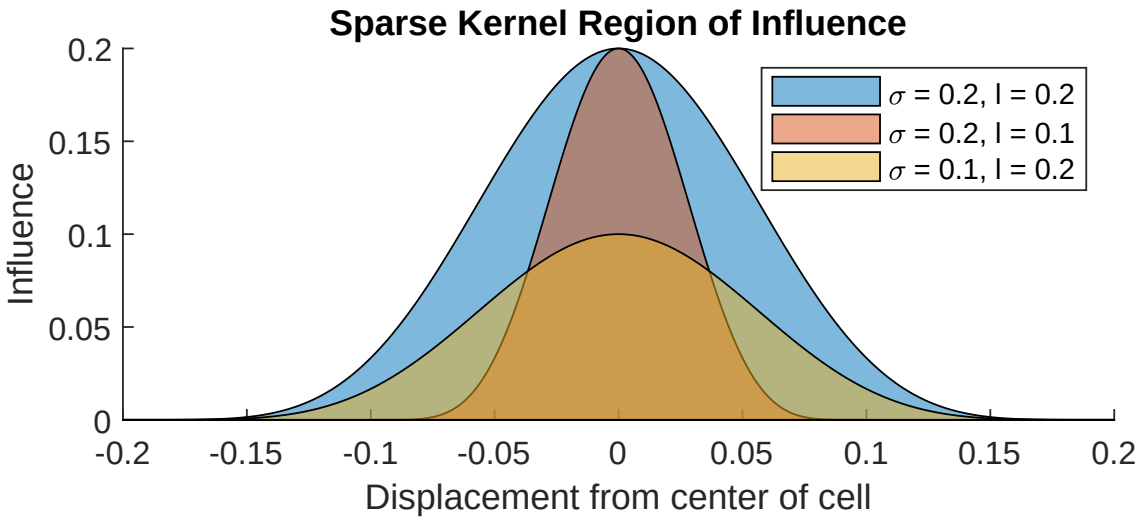



Figure 3.2: An illustrative example of the sparse kernel function (Eq. 3.11) with a variety of values for $\sigma$ and $l$.
The region of influence $R_j$ around a cell $j$ is based on the value of $l$, while $\sigma$ defines the strength of that influence.

influence while $l$ declares the size of the region of influence. As seen in Figure 3.2, when $d$ is small the influence on the predicted occupancy of a map cell will be largest. This procedure for prediction dictated by spatial proximity can improve the accuracy of occupancy map estimation, using relevant data from outside a cell's boundaries to influence its classification.

Equations (3.4) and (3.5) describing the counting sensor model must be updated to incorporate the above kernel function. In those equations, $y_i$ represents the observations of occupancy from the incoming data, such that a hit point will have $y_i = 1$ while a ray passing only through free space will have $y_i = 0$.

$$\alpha_j := \alpha_0 + \sum_{i=1}^{N} k(x_j, x_i) y_i \tag{3.12}$$

$$\beta_j := \beta_0 + \sum_{i=1}^{N} k(x_j, x_i)(1 - y_i) \tag{3.13}$$

Equations (3.12) and (3.13) are used by BGKOctoMap-L and define the influence of range sensor observations on the classification of occupancy map cells. However, there are two other states in our newly proposed variant of this algorithm, unknown and uncertain. Unknown has typically been assumed to apply when $m_j = 0.5 \pm \epsilon$ for some small $\epsilon$, however, the current equation (3.7) for expected value will return $m_j \approx 0.5$ for all instances where $\alpha_j \approx \beta_j$, even when $\alpha_j$ and $\beta_j$ are very large and the state of cell $j$ should be uncertain. Therefore, new classifications will be introduced in BGKOctoMap-LV, both to account for unknown cells, and to separately define cells containing conflicting information as uncertain.

## 3.3 Introducing New Classes into BGKOctoMap-L

### 3.3.1 Defining and Weighting the Unknown Classification

The unknown classification represents regions of a robot's workspace that are yet to be explored. Even when some initial exploration has occurred, a map cell may still be considered partly unknown. Therefore, we propose to incorporate pseudo-evidence into our estimation procedure via a threshold for the unknown classification. Above that threshold, a cell's contents are assumed to be observed, and it must be designated occupied, free or uncertain. To ensure that unknown is only counted when $\alpha_j$ and $\beta_j$ are small, we define a threshold $w_{MIN}$ such that:

$$w_j = \begin{cases} w_{MIN} & \text{if } \alpha_j + \beta_j < w_{MIN} \\ \alpha_j + \beta_j & \text{otherwise} \end{cases} \tag{3.14}$$

The unknown classification is assumed to apply when $m_j = 0.5$, therefore, we will assume that there is pseudo-evidence $\gamma_j$ corresponding to this class, in the same way that $\alpha_j$ accounts for occupied space and $\beta_j$ accounts for free space. Based on our threshold $w_{MIN}$, we can make the assumption that $w_j = \alpha_j + \beta_j + \gamma_j$, which means we can solve for $\gamma_j$ given that we know the other three terms. However, now we need to modify the probability mass function and update the occupancy and variance equations used in our proposed estimation process.

$$P(m_j) = \begin{cases} \dfrac{\alpha_j}{\alpha_j + \beta_j + \gamma_j}, & \text{if } m_j = 1 \\ \dfrac{\gamma_j}{\alpha_j + \beta_j + \gamma_j}, & \text{if } m_j = 0.5 \\ \dfrac{\beta_j}{\alpha_j + \beta_j + \gamma_j}, & \text{if } m_j = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

With this new distribution we can update our estimates for occupancy and the variance of occupancy accordingly.

$$\mathbb{E}[m_j] = \sum_{m_j \in \{0,0.5,1\}} m_j \; p(m_j) = \frac{\alpha_j + 0.5\gamma_j}{\alpha_j + \beta_j + \gamma_j} = \mu_j \tag{3.16}$$

$$\mathbb{V}[m_j] = \sum_{m_j \in \{0,0.5,1\}} (m_j - \mu_j)^2 p(m_j) = \frac{\alpha_j \; \beta_j + 0.25(\alpha_j + \beta_j)\gamma_j}{(\alpha_j + \beta_j + \gamma_j)^2} \tag{3.17}$$

When $\gamma_j = 0$, we can see that Equation (3.16) reverts back to Equation (3.7), which considered only free/occupied classes. Including $\gamma_j$ via the $w_{MIN}$ term will not change the final state of a cell's occupancy, but will produce a result that includes unknown when $\alpha_j + \beta_j < w_{MIN}$. However, this particular equation still allows for the probability of occupancy to be in the unknown range when $\alpha_j \approx \beta_j$, even for large

values. To separate unknown from uncertain, we define a new estimator to push the probability of occupancy to 1 or 0 as $\gamma_j$ shrinks:

$$\hat{\mathbb{E}}[m_j] = \begin{cases} \dfrac{\alpha_j + 0.5\gamma_j}{\alpha_j + \gamma_j} & \text{if } \alpha_j \geq \beta_j \\ \\ \dfrac{0.5\gamma_j}{\beta_j + \gamma_j} & \text{if } \alpha_j < \beta_j \end{cases} \tag{3.18}$$

In typical applications of occupancy grid mapping, the majority consensus of the sensor observations that land within a map cell will dictate whether that cell is labeled free or occupied. Following the same philosophy, Equation (3.18) checks the consensus of past sensor observations, and then pushes the occupancy probability toward the corresponding class. As $\gamma_j$ eventually approaches 0 with the exploration of a robot's workspace, a cell's occupancy probability will approach 1 or 0, indicating that its contents are no longer unknown.

### 3.3.2 Defining the Uncertain Classification

Unfortunately the threshold $w_{MIN}$ does not address the issue that $\alpha_j \approx \beta_j$ should represent uncertainty when they are large, indicating disagreement among sensor observations. There are a few methods capable of dealing with this issue, such as assigning more weight to newer data than to old data [89]. However, our proposed solution allows us to push the probability of occupancy to its upper and lower limits while also allowing the occupancy probability to flip from occupied to free and vice versa without nearing unknown.

We update the variance equation (3.17) based on Equation (3.18), as we need

to use $\mu_j$, which yields:

$$\hat{\mathbb{V}}[m_j] = \begin{cases} \dfrac{(\alpha_j^2 + \alpha_j\gamma_j)(4\beta_j + \gamma_j) + \beta_j\gamma_j^2}{4(\alpha_j + \gamma_j)^2(\alpha_j + \beta_j + \gamma_j)} & \text{if } \alpha_j \geq \beta_j \\ \\ \dfrac{\alpha_j(4\beta_j^2 + 4\beta_j\gamma_j + \gamma_j^2) + \beta_j\gamma_j(\beta_j + \gamma_j)}{4(\beta_j + \gamma_j)^2(\alpha_j + \beta_j + \gamma_j)} & \text{if } \alpha_j < \beta_j \end{cases} \tag{3.19}$$

When $\gamma_j >> \alpha_j, \beta_j$, we can see both forms of variance converge asymptotically to $\frac{1}{\gamma_j}$, which indicates our starting variance when $w_{MIN} >> \alpha_0, \beta_0$. Therefore, if $w_{MIN}$ is sufficiently large, resulting in a large quantity of pseudo-evidence indicating unknown, the starting variance will be small.

However, when $\gamma_j = 0$, the new variance equation simplifies:

$$\hat{\mathbb{V}}'[m_j] = \begin{cases} \dfrac{\beta_j}{\alpha_j + \beta_j} & \text{if } \alpha_j \geq \beta_j \\ \\ \dfrac{\alpha_j}{\alpha_j + \beta_j} & \text{if } \alpha_j < \beta_j \end{cases} \tag{3.20}$$

Known cells will be updated according to this equation. While Eq. (3.20) is different from the traditional variance shown in Eq. (3.8), we can still use this as an uncertainty metric, as it usefully captures the proportionality of the lesser observation count to the total observation count.

We do not apply the "pushing to extremity" premise of Equation (3.18) to a map cell's variance, as that would result in the variance dropping to zero once $\alpha_j + \beta_j > w_{MIN}$, reducing the information we have to describe the occupancy of that cell and our confidence in its prediction.

### 3.3.3 Free Space Representation from Sensor Data

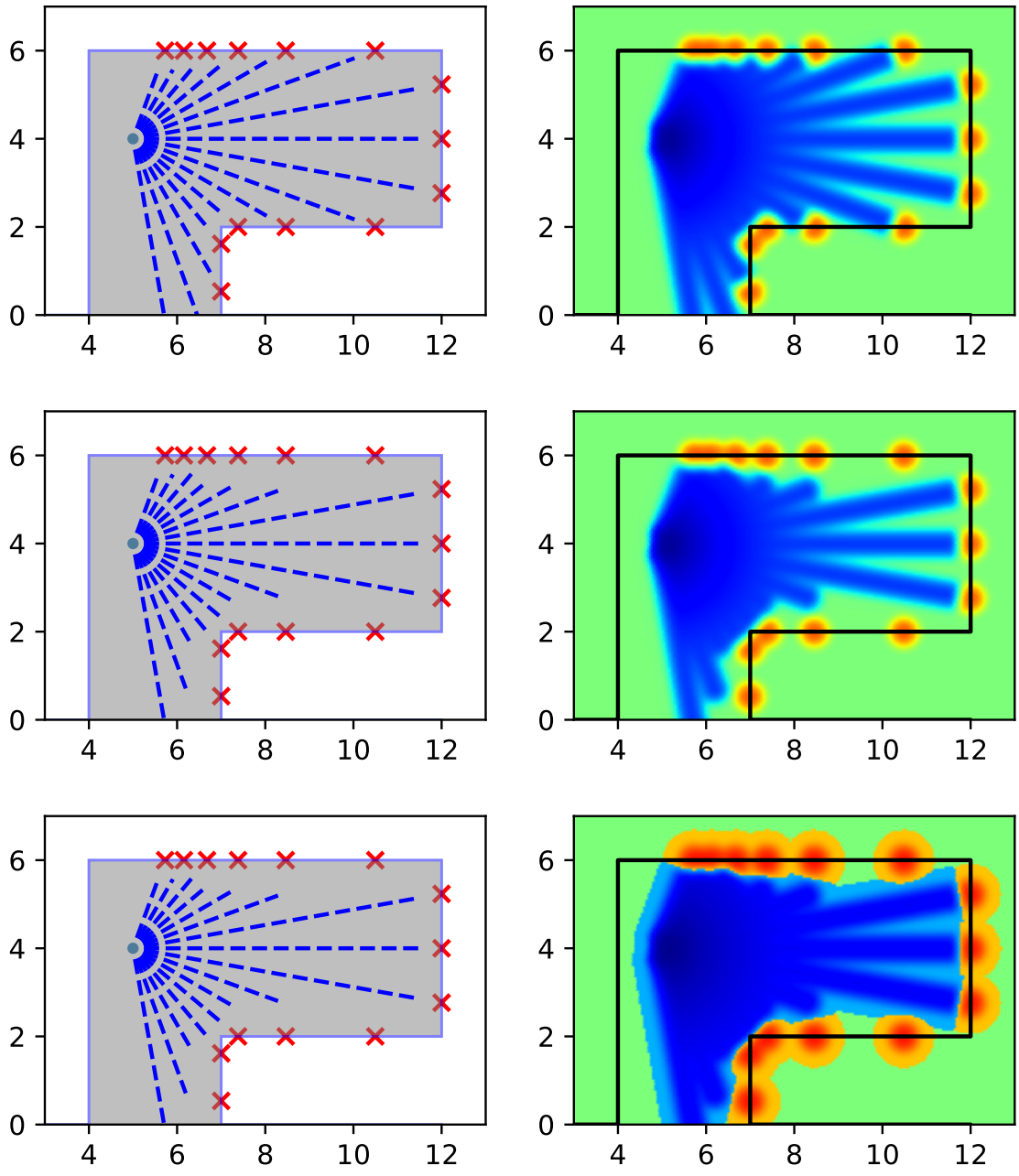



Figure 3.3: An illustrative comparison of occupancy representations. The left shows training data while the right shows the inference result. From top to bottom: (1) point-to-line distance kernel evaluation used in BGKOctoMap-L, (2) reducing the length of glancing rays to minimize overestimation of free space, (3) combining reduced glancing rays and the proposed occupancy predictions from Eq. (3.18).

Rays beginning at the sensor and ending at a hit point are used to define a robot's observations of free space. Due to the nature of range observations radiating at fixed angles from a sensor origin, the free space around the sensor will be more densely represented with data than the surrounding obstacles observed at various hit points. Additionally, with their continuous representation, the free observations along a ray are likely to be closer to a given query point than the single hit point at the end of the ray. Therefore, overestimation of free space occurs if there are no checks in place to avoid it.

In particular, glancing rays, occurring nearly tangential to an obstacle surface, can be a problematic cause of overestimation in predictive occupancy mapping. When using such rays to estimate free space, it is possible to erroneously define free space inside obstacles, as seen at the top of Figure 3.3. On the left of the Figure we can see the true environment, and at right some of the resulting errors introduced into an occupancy map. When rays are nearly tangential to an obstacle wall, part of the wall is assumed to be free space in the resulting map, even with a small standoff from the ray. This is caused by unbiased inference over the contents of the sensor ray, which weighs free space more heavily than occupied space.

To reduce overestimation of free space, the proposed algorithm reduces the length of a ray based on nearby hits. To reduce the ray length, we search within the ray's region of influence for any hits that occur closer to the sensor origin than the hit at the end the ray. If a hit is found, the ray length is shortened to that distance. While this is an exhaustive search, there are a few ways to minimize computational complexity.

By reducing the length of glancing rays passing nearly tangent to nearby obstacles, we are able to minimize the overestimation of free space when the available data is sparse. This procedure complements our proposed Equation (3.18) for updat-

ing occupancy probabilities, which forces occupancy probability toward the free and occupied classes. Figure 3.3 shows the resulting estimation of occupancy probability from a single range scan. At the bottom of Figure 3.3, the border between free and occupied is sharp, well-defined, and can quickly adapt to the arrival of new observations, unlike previous mapping methods. Map variance is not included in this figure, as the map only visualizes the occupancy probabilities.

### 3.4 Algorithm Summary

Two key updates to BGKOctoMap-L have been proposed, and their implementation is discussed in this section. The first reduces the length of glancing rays, augmenting the training data that will be subsequently used to infer occupancy. This method requires the initial training data $\mathcal{X}, \mathcal{Y}$, which contain the positions and occupancy of the raw sensor data. When $y_i \in \mathcal{Y}$ is equal to 1, the corresponding state $x_i \in \mathcal{X}$ is considered occupied, and when $y_i = 0$, the corresponding state is free.

The ray-shortening procedure is summarized in Algorithm 1. To shorten free space rays $\overrightarrow{ox_i}$, which are defined between the sensor origin and a hit, we must compare all rays against all hit points in the current point cloud on which the map is based. After computing the nearest point on the ray to a given occupied point $x_i^{'}$ on line 7, the minimum distance between the ray and point is calculated. If the distance calculated is less than the influence distance $l$, then there is an overlap between the ray and occupied point, which would ordinarily result in overestimation of free space. To mitigate overestimation of free space, the ray is reduced each time a new occupied point is found within its region of influence. This process occurs repeatedly, where the ray and thus its region of influence is updated each time $d < l$, ensuring that the ray is reduced as far as necessary and the search order does not matter.

**Algorithm 3.1:** Reducing Free-Space Training Data

```
   Data: Training data: 𝒳, 𝒴; Origin: o
1  Function trim_training_data(𝒳, 𝒴, o):
2      for each (x_i, y_i) ∈ (𝒳, 𝒴) do
3          if y_i = 0 then
4              Define ox_i ray
5              for each (x_i', y_i') ∈ (𝒳, 𝒴) do
6                  if y_i' = 1 then
7                      Compute x_free from Eq. (3.9) using x_i' as x_*
8                      d = ‖x_free − x_i'‖
9                      if d < l from Eq. (3.11) then
10                         x_i ← reduce_ray(ox_i, x_i')
```

By only changing the input data and the way we process cell data, we are able to continue to use the same inference procedure as BGKOctoMap-L, summarized in Algorithm 2. The objective to compute the posterior parameters given observations $\mathcal{X}$ and $\mathcal{Y}$ remains the same. However, the input data has been altered per Alg. 1, reducing the length of the rays where needed to avoid free-space overestimation. Also unlike BGKOctoMap-L, the posterior data $\alpha$ and $\beta$ are updated per Eqs. (3.14)-(3.19) to predict both occupancy and the variance of occupancy.

As described earlier, the proposed BGKOctoMap-LV algorithm will use Equation (3.18) to solve for expected occupancy and (3.19) to solve for variance. While the equations themselves differ, the structure of the algorithm remains the same as the BGKOctoMap-L algorithm. However, these changes alter the way cell states are defined. Cells are defined as unknown if the expected occupancy lies within the range between the occupied and free thresholds, typically a range of $[0.5 - \epsilon, 0.5 + \epsilon]$. By pushing the occupancy values to their extremes, we can predict the contents of newly discovered cells early and never revert back to unknown. Using Eq. (3.19) for variance, the uncertain state was created by defining a variance threshold. When

**Algorithm 3.2:** Bayesian Generalized Kernel (BGK) Inference

**Data:** Training data: $\mathcal{X}, \mathcal{Y}$; Query point: $x_*$
**Function** BGK($\mathcal{X}, \mathcal{Y}$, $x_*$)**:**
  **for** *each* $(x_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ **do**
    **if** $y_i = 0$ **then**
      Compute $\hat{x_i} \leftarrow x_{free}$ from Eq. (3.9)
    **else**
      $\hat{x_i} \leftarrow x_i$
    $k_i \leftarrow k(x_*, \hat{x_i})$ Sparse kernel, Eq. (3.11)
    $\alpha_* \leftarrow \alpha_* + k_i y_i$
    $\beta_* \leftarrow \beta_* + k_i(1 - y_i)$
  **return** $\alpha_*, \beta_*$

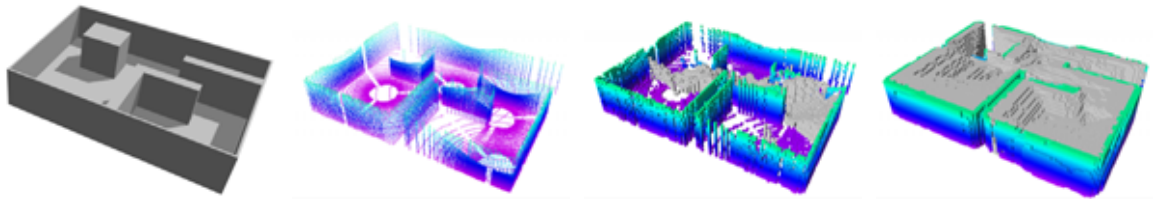
the variance is above that threshold, regardless of expected occupancy, the state is considered uncertain.

### 3.5 Experimental Results

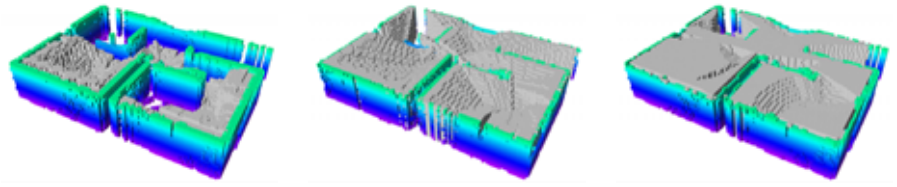
Using OctoMap [9] as a baseline, we compare our proposed algorithm, BGKOctoMap-LV[1], against other inference-based maps including Gaussian Process OctoMaps (GPOctoMap) [13], the original learning-aided Bayesian generalized kernel map (BGKOctoMap-L) [12], and a newer framework for adaptive kernel inference (AKIMap) [86]. While these algorithms differ in many ways, a few key components were kept constant throughout our tests. The range of *known* occupancy states was separated into two regions, known free and known occupied. Known free was declared to be the region of occupancy where $p = \hat{\mathbb{E}}[m_j] \leq 0.3$ while the known occupied region was defined by $0.7 \leq p = \hat{\mathbb{E}}[m_j]$ for all algorithms.

GPOctoMap, BGKOctoMap-L and our proposed variant BGKOctoMap-LV all use variance thresholds to define unknown and uncertain occupancy states. Due to

[1]The code for this paper is implemented in a branch of the Learning-aided 3D Mapping Library, available at `https://github.com/RobustFieldAutonomyLab/la3dm/tree/feature/bgklv`.

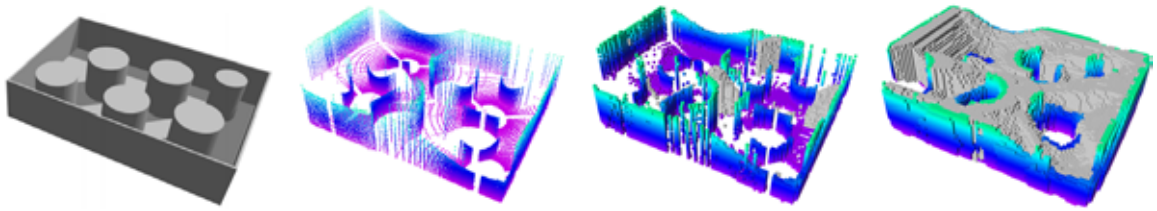


(a) Environment Model (b) Point Cloud (c) OctoMap (d) GPOctoMap

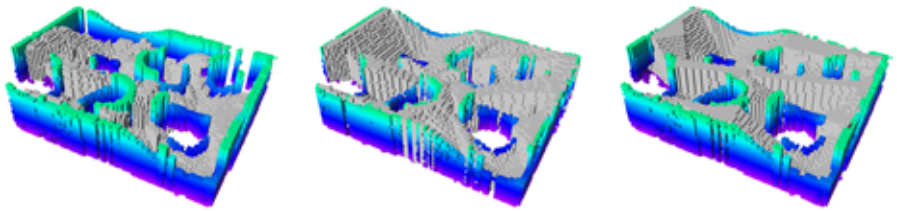


(e) BGKOctoMap-L (f) AKIMap (g) BGKOctoMap-LV

Figure 3.4: Results from using the same Structured map used in [12] and [86] to test mapping inference on rectangular obstacles.



(a) Environment Model (b) Point Cloud (c) OctoMap (d) GPOctoMap



(e) BGKOctoMap-L (f) AKIMap (g) BGKOctoMap-LV

Figure 3.5: Results from using the same Unstructured map used in [12] and [86] to test mapping inference on curved obstacles.

the uniqueness in how variance is treated among each separate algorithm, they all required different values for their thresholds. The values used for our tests can be found in Table 3.1. A few other parameters were kept constant during all our testing for BGKOctoMap-L and the proposed BGKOctoMap-LV. Namely, the beta prior

parameters $\alpha_0$ and $\beta_0$ were assigned very small values to assume virtually no prior information about the environment. To ensure fairness in the results and follow the same process, $w_{MIN}$ was assigned a very small value as well, resulting in minimal usage of the unknown class when evaluating the accuracy of map predictions.

| Parameter | Description | Value | Algorithms |
|---|---|---|---|
| occ_thresh | Occupancy Threshold | 0.7 | All |
| free_thresh | Free Threshold | 0.3 | All |
| var_thresh | Variance Threshold | 0.02 | GP |
| | | 0.15 | BGK-L |
| | | 0.2 | BGK-LV |
| $\alpha_0, \beta_0$ | Beta Prior Parameters | 0.001 | BGK-L/BGK-LV |
| $w_{MIN}$ | Unknown Prior Parameter | 0.001 | BGK-LV |

Table 3.1: List of parameters which are the same in all tests performed

Similar to past comparisons [12, 13, 14], two simulated datasets were utilized with known ground truth to check each algorithm for accuracy. The first environment was "structured" with predominantly rectangular features, while the second environment was "unstructured" and contained cylindrical objects surrounded by a rectangular perimeter. Both of these datasets have virtually no overlapping point cloud data, which challenges inference-based algorithms using sparse data.

However, accuracy is not the only way to compare inference-based mapping algorithms. Rather, how effective the inference-based algorithms are at parsing sparse data into a reasonable representation of the environment should also be checked. To do so, we used data captured from our own Jackal ground robot of the Strataspace mine in Louisville, KY using a VLP-16 Lidar. Lidar sensors result in very dense data, therefore to rigorously compare inference capabilities, we artificially sparsified the data by downsampling the incoming point clouds.

Each test was performed on a desktop with an 8-core 3.60 GHz Intel i9 CPU with 16 threads running Ubuntu Linux. Every algorithm was run via the Robot

Operating System (ROS) [6] and utilized the Point Cloud Library (PCL) [5].

### 3.5.1 Simulated Data

Both the structured and unstructured environments are $10.0 \times 7.0 \times 2.0$ m in size, with twelve non-overlapping range scans provided as input. Only the cells that are seen in Figs. 3.4 and 3.5 were used to evaluate the accuracy of each algorithm. The colored cells indicate the known occupied state while the gray cells indicate the known free state. Both unknown and uncertain cells were not considered for the purpose of plotting the receiver operating characteristic (ROC) curves, to accurately capture this mapping scenario as a binary classification problem.

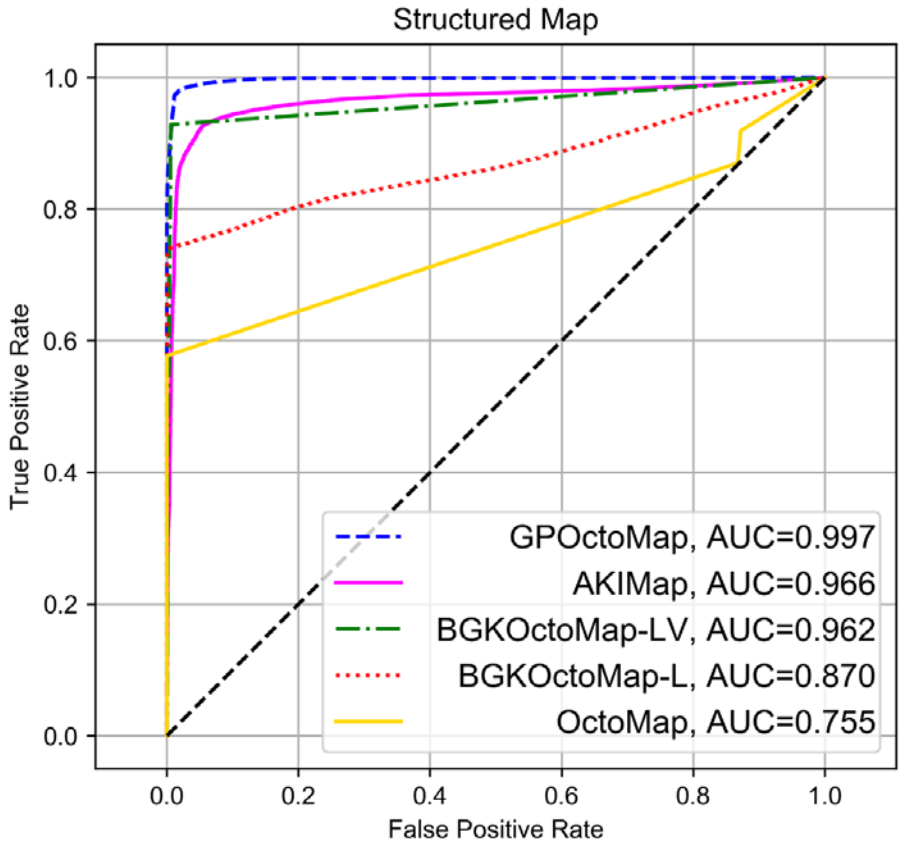



Figure 3.6: ROC curves for the Structured Map

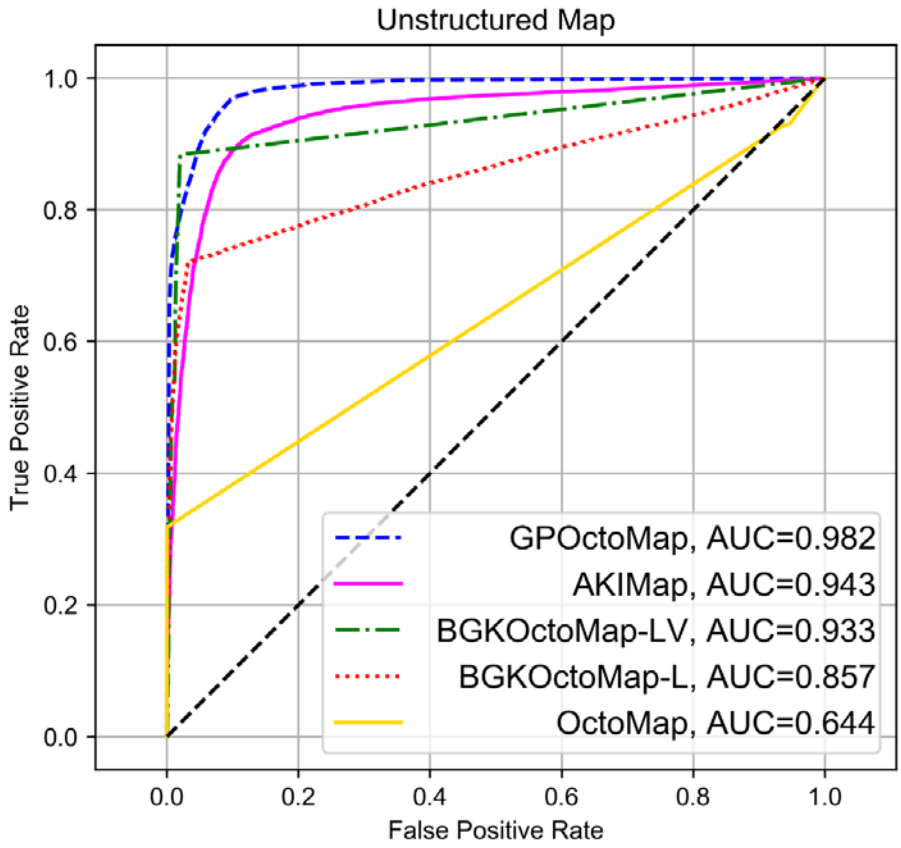



Figure 3.7: ROC curves for the Unstructured Map

To ensure consistency in the data, each algorithm used 0.1m resolution without any pruning. Various other parameters used specifically for this simulated mapping test can be found in Table 3.2. Each of the parameters used was either tuned to approximately optimize the performance of each algorithm, or were defined by their original authors as the default values. In particular, the free sample resolution used by BGKOctoMap-L was set to maximize the known free cells, while minimizing the losses incurred by the known occupied cells when the data conflicts. Both the GPOctoMap and BGKOctoMap-LV algorithms were able to use any value less than or equal to the kernel length values for their free sample resolutions. The kernel length and scale values used were the default values for that respective resolution. As the

BGKOctomap-LV algorithm was based on BGKOctomap-L [12], we used the same default values for our kernel length and scale. These were chosen based on previous experiments, where larger kernel lengths led to excessive homogeneity within the maps, while the scale changed how quickly the unknown cells disappear.

| Parameter | Description | Value | Algorithms |
|---|---|---|---|
| resolution | Resolution | 0.1m | All |
| ds_resolution | Down-sample Resolution | 0.1m | All |
| block_depth | Block Depth | 1 | All |
| free_resolution | Free Sample Resolution | 0.1m | GP/BGK-LV |
| | | 0.85m | BGK-L |
| $l$ | Kernel Length | 1.0m | GP |
| | | 0.2m | BGK-L/BGK-LV |
| $\sigma_0$ | Kernel Scale | 1.0 | GP |
| | | 0.1 | BGK-L/BGK-LV |

Table 3.2: List of parameters used in the simulated mapping tests

Receiver operating characteristic (ROC) curves were used to check each algorithm for its accuracy with respect to each environment. Only the known free and known occupied cells shown in Figs. 3.4 & 3.5 were used in the ROC curves, resulting in Figs. 3.6 & 3.7 respectively. The ROC curve of the structured environment shows how the proposed algorithm BGKOctoMap-LV has comparable accuracy to new techniques such as AKIMap and makes improvements on the prior BGKOctoMap-L. The same conclusion can be drawn from the ROC curve describing the unstructured environment. Thus we believe that the accuracy of the proposed algorithm is sufficient for use in the field.

### 3.5.2 Lidar Mapping Experiments

We collected VLP-16 lidar data from the Strataspace mine in Louisville, KY to compare predictive mapping algorithms[2]. The subset of the data we selected for these

[2]A video showing the process of each mapping algorithm performing on this data is available at `https://youtu.be/VTUc4lQ2en4`.

| Parameter | Description | Value | Algorithms |
|---|---|---|---|
| resolution | Resolution | 0.2m | All |
| ds_resolution | Down-sample Resolution | 0.5m | All |
| max_range | Maximum Sensing Range | 30m | All |
| block_depth | Block Depth | 4 | GP |
| | | 5 | BGK-L |
| | | 6 | BGK-LV |
| free_resolution | Free Sample Resolution | 0.1m | GP/BGK-LV |
| | | 6.5m | BGK-L |
| $l$ | Kernel Length | 1.0m | GP |
| | | 0.6m | BGK-L/BGK-LV |
| $\sigma_0$ | Kernel Scale | 1.0 | GP |
| | | 0.1 | BGK-L/BGK-LV |

Table 3.3: List of parameters used in the mine tests

experiments resulted in a $130 \times 130 \times 7$ m environment. To necessitate the use of predictive mapping, the point cloud data was sparsified, downsampling the original scans to 0.5m leaves. Without downsampling, the data was so dense that all maps performed similarly.

While map accuracy is important, it often ignores some defining features of a "good" map. For instance, a map may have very high accuracy, but only define two or three cells as occupied or free. Other factors need to be taken into account when considering what defines a "good" map, such as coverage. In a live experiment, point cloud data is often gathered very quickly and not every scan can be utilized if a mapping algorithm wishes to perform in real-time. However, not all scans are necessary if there is sufficient overlap. Therefore the frequency at which a map updates is important to mapping effectively. Another useful metric is how much pruning has occurred. Funk et al. in [11] prove that successful pruning can be used to speed up motion planning. When more pruning occurs, fewer cells are used to cover the same volume.

The data structure used by GPOctoMap, BGKOctoMap-L and BGKOctoMap-

| Algorithm | Block Depth | Total Coverage ($m^3$) | Average Cell Size | Average Update Frequency (Hz) |
|---|---|---|---|---|
| OctoMap | N/A | **38382** | **0.0296** | **3.27** |
| Akimap | N/A | **35364** | **0.0080** | **2.24** |
| GPOctoMap | 1 | 28827 | 0.0080 | 0.91 |
| | 2 | 37083 | 0.0434 | 2.39 |
| | 3 | 42699 | 0.0872 | 3.73 |
| | **4** | **43749** | **0.0806** | **2.63** |
| | 5 | 18034 | 0.0259 | 0.07 |
| | 6 | 1031 | 0.0144 | <0.01 |
| BGKOctoMap-L | 1 | 7862 | 0.0080 | 1.19 |
| | 2 | 18153 | 0.0253 | 3.32 |
| | 3 | 25229 | 0.0331 | 5.54 |
| | 4 | 31386 | 0.0407 | 5.05 |
| | **5** | **35956** | **0.0488** | **2.73** |
| | 6 | 32159 | 0.0444 | 0.59 |
| BGKOctoMap-LV | 1 | 33796 | 0.0080 | 0.13 |
| | 2 | 38451 | 0.0342 | 0.61 |
| | 3 | 38928 | 0.0508 | 0.86 |
| | 4 | 38932 | 0.0536 | 0.83 |
| | 5 | 38911 | 0.0543 | 0.72 |
| | **6** | **38326** | **0.0544** | **0.51** |

Table 3.4: Varying block depth to decide which to use for lidar mapping experiments

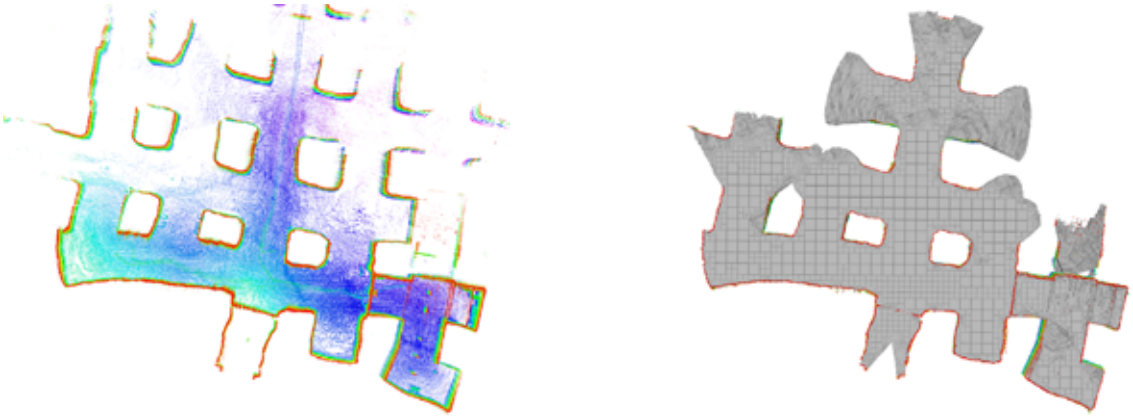
LV has limitations on its pruning capabilities. "Blocks" are defined as a set of neighboring cells, where block depth is a parameter used to define how many cells exist within a given block. Each increase in block depth is equivalent to another level of possible pruning, however there is also increased computational complexity as block depth increases. Due to the direct link between block depth and how much pruning can occur, each algorithm was tested at differing block depths over the same subset of the mine data used to compare all the algorithms later. Comprehensive testing resulted in the data shown in Table 3.4, where the largest feasible block depth is emboldened and used in the final experimental results. The lower update frequencies at block depths one and two are a result of being unable to fully utilize the multi-threaded functionality of each algorithm at those depths.

Besides block depth, which was varied for these experiments, the remaining

parameters were kept constant; all values are summarized in Table 3.3. The resolution was set to 0.2m due to the size of the environment used. That resolution size resulted in our choice for down-sample resolution, which affected the kernel characteristic length used by BGKOctoMap-L and BGKOctoMap-LV. The kernel length dictates how far away from a cell we should consider data to be influential, thus the kernel length must be larger than the down-sample resolution. However, the value chosen for the two BGKOctoMap algorithms was still not as large as the one used by the GPOctoMap algorithm, which was left at the default value.

OctoMap and AKIMap do not explicitly use block depth, thus they were parameterized separately. AKIMap has no pruning system in place and was left that way for these tests. OctoMap does have a pruning function, but it can perform that function without the data structure built via block depth used by the GP and BGK algorithms, which allowed OctoMap to heavily prune free space in this experiment without depth parameterization.

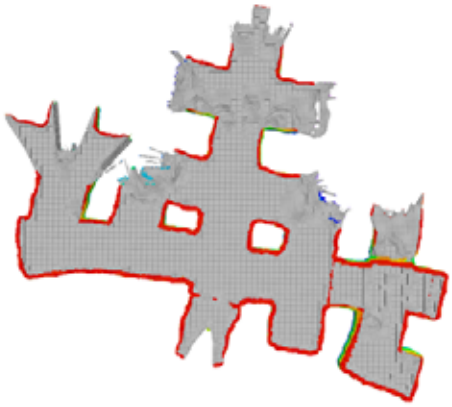
Coverage is the sum total of the volume defined by known free and known occupied cells. Average cell size is the coverage divided by the total number of cells defined by known free and occupied. The average update frequency was calculated by finding the total number of maps published by each algorithm over the course of the dataset, and dividing by the full duration of the dataset. While all algorithms were fastest at block depth three, more pruning occurred at larger block depths. Therefore, the metric for choosing which block depth to use for each algorithm was based primarily on total coverage. GPOctoMap had the most coverage at block depth four, while BGKOctoMap-L had the most coverage at block depth five. The coverage decreased when the algorithms were too slow to process enough lidar scans, resulting in missed data. BGKOctoMap-LV used block depth 6 due to the increase in average cell size with a minimal loss of coverage.



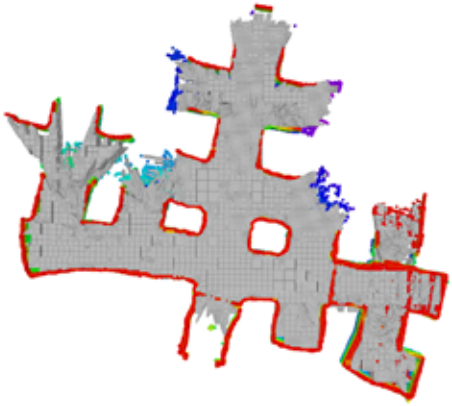
(a) Point Cloud



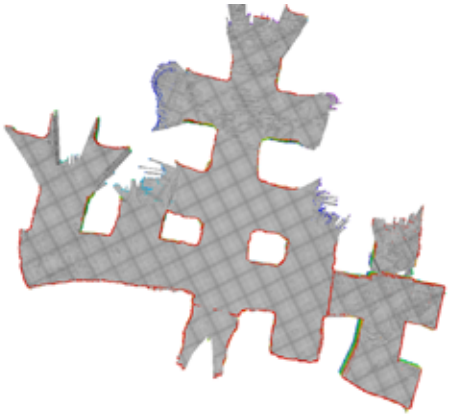
(b) OctoMap



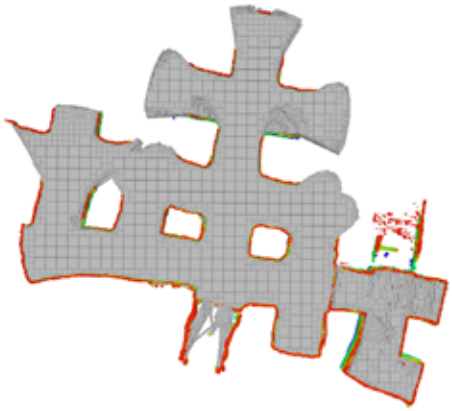
(c) GPOctoMap



(d) BGKOctoMap-L



(e) AKIMap



(f) BGKOctoMap-LV

Figure 3.8: Each mapping algorithm was tested using the same lidar dataset collected from an underground limestone mine. Lidar odometry was employed while mapping to minimize drift. To help visualize the data, we only show data below z = 3m. Gray cells indicate free space, while colored cells indicate occupied space. The color change of occupied cells indicates their height.

Most of these algorithms operated successfully around the 2-3Hz range while the proposed algorithm managed to succeed at $> 0.5$Hz. While more processing is required, BGKOctoMap-LV achieved accurate inference over the incoming data. Our proposed algorithm did not see much of a decrease in processing time as block depth increased, because incoming data was processed using a different method which is no longer an exponential function of block depth, which differs from previous algorithms. GPOctoMap and BGKOctoMap-L used the size of a block to partition incoming data into sets, which became superfluous at larger block depths. For smaller block depths, that system improved processing speeds, which was necessary for GPOctoMap to run in real-time by utilizing multi-threading capabilities. By comparison, the proposed BGKOctoMap-LV algorithm defines sets of incoming data for each map cell, regardless of block size, which allows us to successfully utilize larger block depths. The difference can be seen by the drop from 2.63Hz to 0.07Hz as GPOctoMap goes from block depth four to five. Similar behavior occurs as BGKOctoMap-L goes from block depth five to six, where the average update frequency drops from 2.73Hz to 0.59Hz. Our proposed algorithm, BGKOctoMap-LV, does not change as drastically as block depth increases, with the worst case from block depth five to six resulting in a 0.21Hz decrease in processing speed.

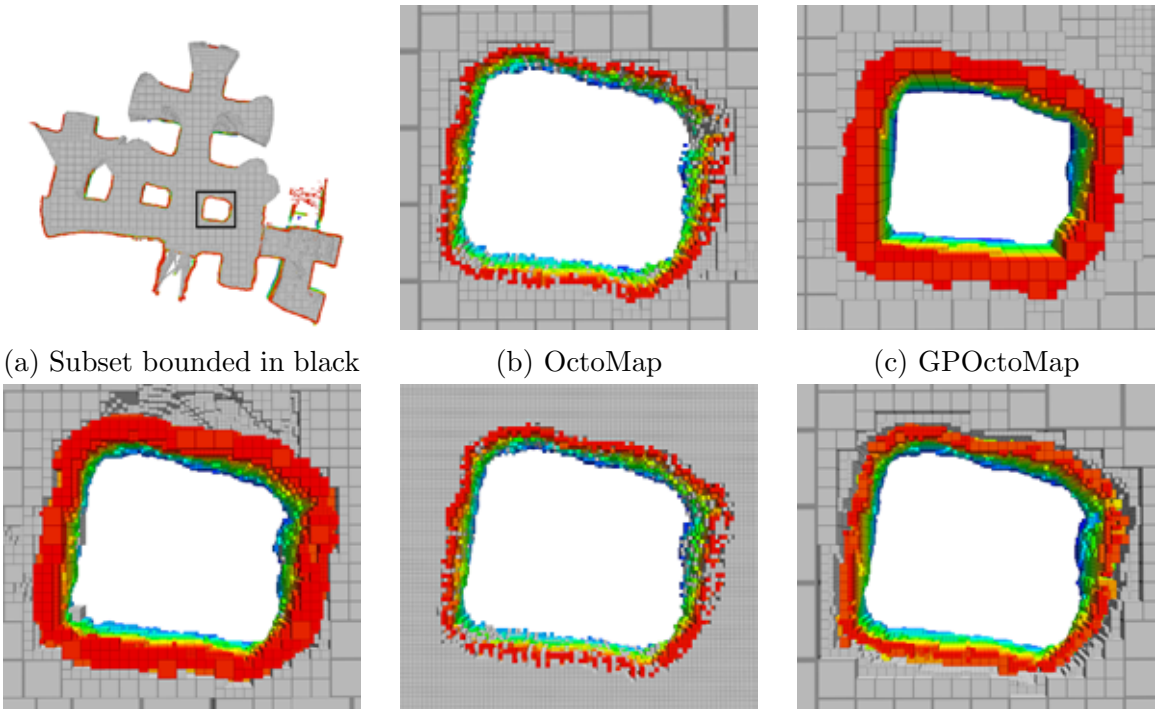
While GPOctoMap produced a very large average cell size at both block depth three and four, the data may be misleading. GPOctoMap performed with strong inference capabilities, producing a much smoother representation of the environment. When such smoothing occurs, very few smaller cells remain as most are pruned, being merged together with their similar neighbors. This behavior actually reduces the accuracy of certain real-world features, such as bumpy surfaces or protrusions. However, those same features would decrease the average cell size of a completely accurate occupancy grid map. Therefore, even though it seems GPOctoMap is vastly

outperforming at lower block depths, the map produced is likely to under-perform in path planning exercises compared to the larger block depths used by BGKOctoMap-L and BGKOctoMap-LV.

Using the parameters shown in Table 3.3, each algorithm was tested over the data visualized in Fig. 3.8a. The point cloud data was not range-limited, showing more than what was utilized by each mapping algorithm. Color is used to indicate height, with red indicating the top and blue the bottom. For ease of visualization, each point was given a non-zero size. All of the images in Fig. 3.8 were truncated at a height of 3m to better illustrate how each algorithm inferred free space. However, when testing each block depth, we did not impose a cutoff height.

Each algorithm accurately captured the global characteristics of the mine, with some aliasing occurring in the AKIMap image due to its lack of pruning. By mapping everything at the finest available resolution, AKIMap produced thin obstacle boundaries and far too many map cells to permit rapid path planning. OctoMap performed one task that was implemented into the proposed BGKOctoMap-LV algorithm but does not exist in the inference-based competitors, which is utilizing rays up to max range from obstacles observed beyond the max range. By utilizing every ray, OctoMap was able to successfully fill in free space quite effectively even with sparse data. GPOctoMap was able to capture the majority of free space with the largest cell size available at the respective block depth used. This is due to its aggressive inference capabilities, which also resulted in an inflated representation of obstacle boundaries. BGKOctoMap-L also resulted in inflated obstacle boundaries, however we believe the cause to be different. The free resolution of BGKOctoMap-L used for this experiment was quite large, at 6.5m. Free space would be significantly over-represented if it was not directly curbed. By using such a large value, we were able to reduce the overestimation of free space while still acquiring an accurate map, which

can be seen in Figure 3.8d. However, a consequence of curbing free space was the overestimation of occupied space.



(a) Subset bounded in black (b) OctoMap (c) GPOctoMap

(d) BGKOctoMap-L (e) AKIMap (f) BGKOctoMap-LV

Figure 3.9: For closer inspection, we chose a subset of the mine data that represents the qualitative performance of each mapping algorithm with our predetermined parameters, showing how each map defines occupied space, free space, and the boundary between them. Too much inference results in overestimation, while too little results in an incomplete representation of the wall.

The proposed BGKOctoMap-LV produced a very rich and well-pruned free space representation of the mine data. However, this result is not unique to BGKOctoMap-LV. Therefore, we will zoom in on a subset of the map to better understand the boundary between occupied and free cells. In Fig. 3.9, we inspect a single pillar within the mine that enables us to more clearly visualize the differences between all algorithms. OctoMap has gaps in the occupied data due to the lack of inference applied to the range data. AKIMap performed similarly, likely due to limits on the range of influ-

ence of sensor data, which was based on its fine resolution. Both GPOctoMap and BGKOctoMap-L had similarly inflated representations of occupied space, however, GPOctoMap added thickness atop free space observations, while BGKOctoMap-L expanded into the free space due to pulling the free space back from the walls. The proposed algorithm BGKOctoMap-LV resulted in well-defined free space and accurate estimation of occupied space. There are some small gaps in the data between known free and known occupied cells due to the uncertainty caused by high variance, however those gaps would be at most a min-resolution cell thick. The proposed algorithm is able to accurately represent the unambiguously free and occupied regions of the workspace, while capturing uncertainty on the boundary between them.

Reducing the inflated representation of occupied space in BGKOctoMap-LV may have unexplored benefits. Uneven surfaces, such as those caused by furniture or certain wall features, are expected to be captured accurately with less generalization. One example to consider is a thin wall that is seen from both sides. In the BGKOctomap-L case, we would expect the wall to be estimated significantly thicker, as overestimation of occupancy occurs from the wall surface toward the sensor on each side of the wall. GPOctomap pushes the overestimation behind the surface, resulting in a more accurate yet still excessively thick wall, where newer data may fix the currently viewed side of the wall at the cost of expanding the reverse side. BGKOctomap-LV minimizes this overestimation, resulting in the most accurate representation of a thin wall seen from both sides. Thin walls are very common within indoor environments, which indicates the frequency and potential relevance of this scenario.

## 3.6 Conclusion

In this section, we have proposed BGKOctoMap-LV, enhancing the BGKOctoMap-L inference-based occupancy grid mapping algorithm to accurately capture known free, known occupied, unknown, and uncertain states. The framework's accuracy was confirmed using simulated mapping scenarios, comparing against recently proposed algorithms in inference-based occupancy grid mapping, while real lidar data was also used to qualitatively illustrate their different outcomes intuitively.

We expect a future version of this algorithm to prove useful for exploration and inspection in 3D environments. Separating *uncertain* from *unknown* will allow us to develop new methods of exploration using inference based mapping that focus exclusively on the *unknown* classification and disregard those map cells considered *uncertain*, for the purpose of efficiently completing the exploration mission. A different approach could utilize both states for a two step exploration and interrogation method. With the ability to drive down uncertainty without depending on the unknown classification, mobile robot exploration algorithms could interrogate structures until they have been completely observed, before continuing to explore.

# Chapter 4

# Simplified Multi-Session Robot Navigation using Waypoints

## 4.1 Problem Description

Mobile robots are often programmed for repeatable tasks, and each instance typically requires the same code. However, repeatable tasks require consistency between attempts, and localization is an important contributing factor to this consistency. For unmanned ground vehicles (UGVs) like Clearpath's Jackal seen in Fig. 4.1, reliable navigation to specified waypoints can facilitate a wide range of repeatable tasks. For example, mobile manipulator platforms would be able to perform repeatable grasping and manipulation tasks at specified locations. However, there are currently no simple, publicly available localization methods and implementations compatible with repeated waypoint navigation that incorporate fast map construction from scratch.

## 4.2 Architecture Description

To ensure accurate waypoint navigation, a global reference map and some method of performing localization are required. Global reference maps can be composed of labeled data, such as landmarks, or raw metric data, such as a point cloud. Many localization methods rely on a specific type of global reference map to perform optimally. SLAM algorithms build global reference maps from scratch.

Our framework uses publicly available packages to create a global reference map from a prior manual expedition, followed by localization for waypoint navigation in multi-session tasks. The package we selected for creating a global reference map


Figure 4.1: Unmaned Ground Vehicle from Clearpath Robotics called Jackal.

was *hdl_graph_slam*[1] [90] for two main reasons. The SLAM results were successful even when only LiDAR point cloud data was given, which reduced the requirements for data collection. There is also a service included that allows users to save a copy of the completed global map in the correct data type for the subsequent localization package.

The real-time localization package we chose is a publicly available Iterative Closest Point (ICP) [91] implementation from ETH Zurich[2]. With some minor modifications for our specific systems, their package was able to take an initial position for the sensor used and accept odometry data for an estimate before performing ICP localization between an active sensor and the global reference map. With further testing, this localization method performed well in cases where the sensor had partial

[1]https://github.com/koide3/hdl_graph_slam
[2]https://github.com/leggedrobotics/icp_localization

occlusion, dynamic obstacles moved around, and even when portions of the global reference map had been changed, such as moved boxes or chairs.

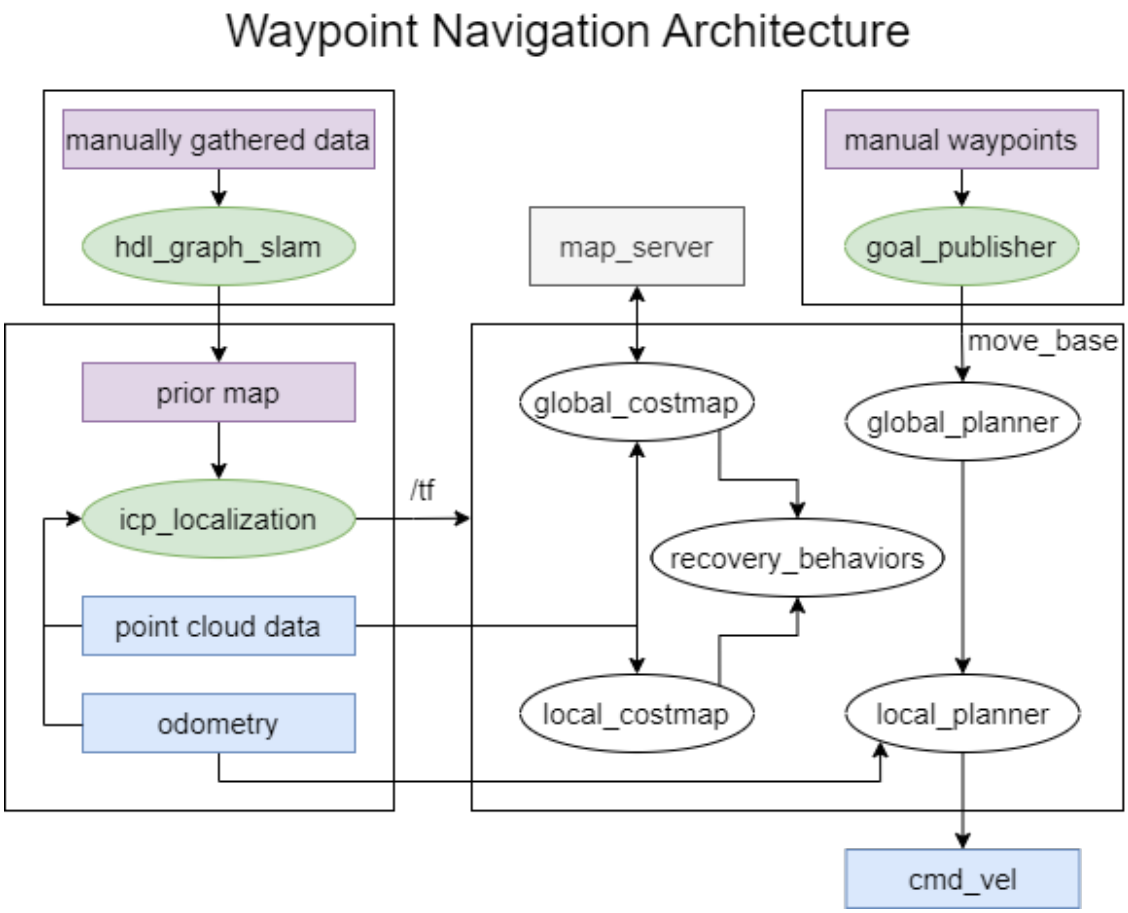



Figure 4.2: Complete architecture of the proposed waypoint navigation package. Purple boxes represent data that only needs to be handled once. Blue boxes are sent and received by the robot. Green ellipses show the new packages used by this architecture.

To enable movement with the localization package, a navigation stack was implemented. The default 2D navigation stack in ROS is the *move_base*[3] package. The full navigation stack includes obstacle avoidance with 2D costmaps and computes collision-free paths from the current position to a global goal using Dijkstra's algorithm [92] by default. Once a goal has been set, the navigation stack sends

[3]http://wiki.ros.org/move_base

velocity commands to the robot until the goal has been reached. The new *waypoint_navigation*[4] package we have published includes the complete waypoint navigation architecture shown in Fig. 4.2, with simulated and real-world example launch files using a Jackal ground vehicle with a mounted LiDAR. Included in our integration of the packages shown in Fig. 4.2 is a waypoint publisher script. The current version of that script accepts 2D goal positions as parameters in a .yaml file, before creating a list of goal positions. When the node is activated, the goals are published in order while waiting for the robot to arrive at its current goal. Once the robot has successfully arrived, the next goal is published until no further goals remain on the list. While simple, this script is effective at supporting repeated autonomous navigation to a series of waypoints.

## 4.3 Experiments

Multi-session localization for waypoint navigation requires accuracy and precision. Accuracy can be defined by how close the robot is to the target, while precision measures the consistency for a given target. As we are using a ground vehicle to test our localization algorithm, the waypoints will be defined by their 2D Cartesian coordinates and a quaternion for orientation, $w_i = (x_i, y_i, q_i)$ for the $i$th position of an ordered list $\mathcal{W}_n = (w_1, w_2, \ldots, w_n)$ of $n$ waypoints. The Jackal UGV used in our work is described by its pose $s_\tau = (x_\tau, y_\tau, q_\tau)$ at time $t = \tau$.

### 4.3.1 Simulation Comparisons

The proposed waypoint navigation architecture of Fig. 4.2 was built to be simple to use and quick to implement. Given the lack of existing localization algorithm implementations that incorporate navigation, the only practical competitor was SLAM,

[4]https://github.com/RobustFieldAutonomyLab/waypoint_navigation

as implemented to support waypoint navigation in the ROS navigation stack. Many SLAM algorithm implementations exist within ROS, however the simplest to use is GMapping [93], the default SLAM framework. Therefore, comparisons were made with respect to accuracy and precision of waypoint navigation for both GMapping, and ICP localization (as implemented within our architecture). Simulations were performed in ROS Noetic with an i9-9900K 3.60GHz CPU and 64GB of RAM. ICP trials were given 5 minutes to reach completion, while GMapping required 6 minutes per trial.

Extensive simulations were performed in Gazebo to compare our proposed use of ICP localization within the architecture of Fig. 4.2 against the conventional usage of GMapping[5] for waypoint navigation within ROS. While a simulated Jackal UGV manages its state estimation process, the Gazebo simulation environment can provide ground truth. Therefore, to test for accuracy and precision we recorded the ground truth state of the robot when navigation to each designated waypoint was completed. If we denote the time when the robot arrived at waypoint $i$ as $\tau i$, then all our ground truth data for a single trial forms the set $\mathcal{S}_n = \{s_{\tau 1}, s_{\tau 2}, \ldots, s_{\tau n}\}$.

[5]https://wiki.ros.org/gmapping

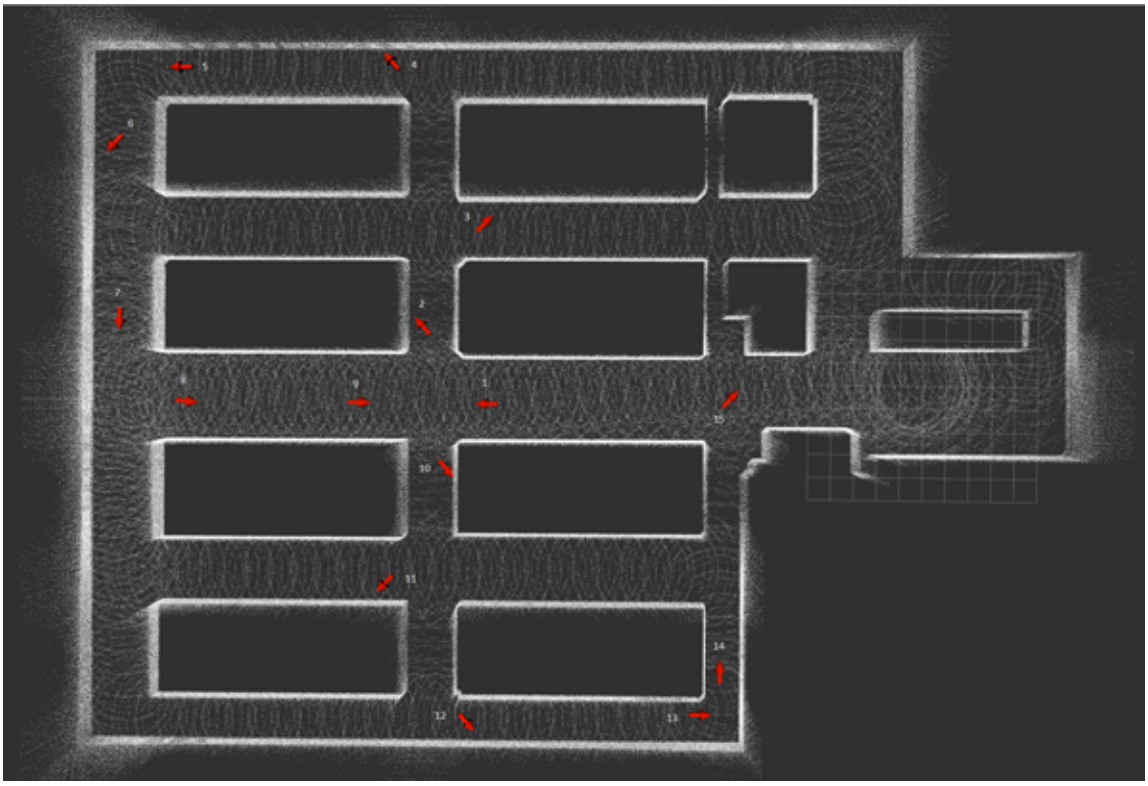


Figure 4.3: Simulated Gazebo environment with 15 marked waypoints in red. Each cardinal and inter-cardinal direction is represented at least once. The white point cloud is the global reference map used for ICP localization.

A simplified tunnel-like map seen in Fig. 4.3 was used for testing with $n = 15$ waypoints chosen to reflect a variety of positions and orientations. Each trial used the same 15 waypoints assigned in the global reference frame. To ensure a sufficiently large dataset, $m = 200$ trials were performed for each waypoint. However, not every trial was successful, resulting in less than 3000 individual datapoints. If failures occurred in the middle of a trial, data was collected up to the last successfully navigated waypoint. Therefore, we were able to compute a success rate in addition to the accuracy and precision of each framework.

Multi-session waypoint navigation implies a first session followed by many subsequent sessions. The initial session can be performed by starting a robot in the ideal

**Algorithm 4.1:** Simulating 200 trials of navigation to 15 waypoints

```
   Data: W_15
 1 m ← 200
 2 count ← 0
 3 n ← 15
 4 for count < m do
 5 |   begin localization simulation
 6 |   i ← 0
 7 |   for i < n do
 8 |   |   navigate to w_i ∈ W_15
 9 |   |   record robot state
10 |   |   i++
11 |   end simulation
12 |   count++
```

state, with no errors. However, under autonomous navigation, the robot needs to navigate itself back to the home position and orientation. Any errors accumulated during navigation will affect future sessions if not accounted for. Therefore, we performed two phases of simulation, in which the first phase represents an initial session where the robot was placed exactly at the origin of the map for each trial. By the comparing the final UGV ground truth locations against the assigned waypoints, we created a set $\mathcal{E}_n = \mathcal{S}_n - \mathcal{W}_n = \{e_1, e_2, \ldots, e_n\}$ of target errors. These errors are plotted in Fig. 4.4 and show how GMapping resulted in more accurate waypoint navigation. However, it is clear that the ICP framework has higher precision, as the convex hulls around each individual waypoint's error $e_i$ form relatively small clusters.

The second phase of simulation is conducted similarly to the first, with the only change being the starting location of the robot. For these trials, a random error from the first session $e \in \mathcal{E}$ was added to the robot's initial pose. The same map, set of waypoints $\mathcal{W}_n$, and number of trials were performed as in the first round of simulation. However, the results show significantly worse errors for the GMapping

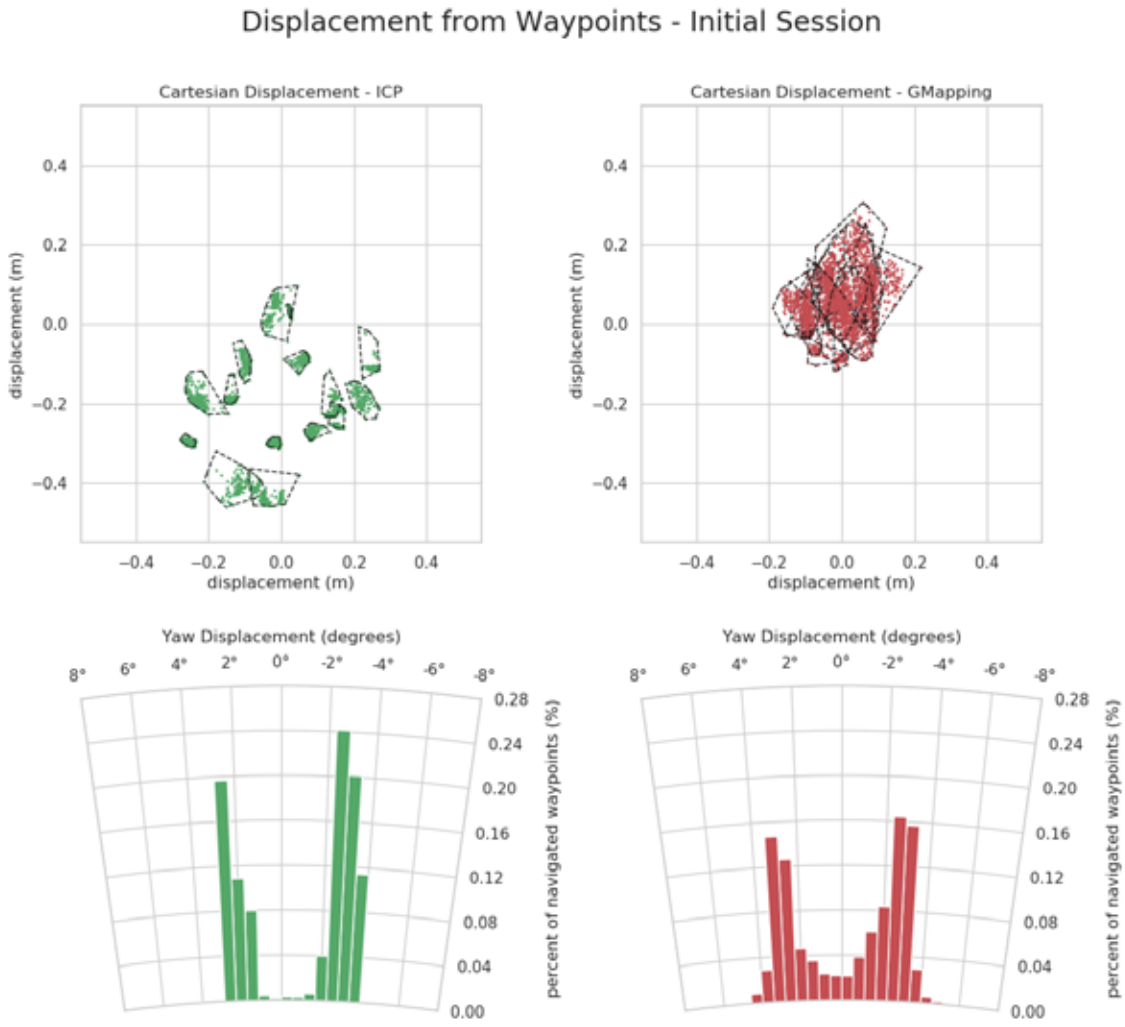



Figure 4.4: Translational and rotational displacements of Jackal UGV when navigating to waypoints in simulation. Convex hulls around data from each waypoint illustrate navigation precision within the translational displacement plots at top.

framework, while the ICP framework barely had any changes, as seen in Fig. 4.5. The success rate of each method was also affected during the change from initial session to second session. For the initial session, ICP had 198 trials reach the first goal point successfully, while only 134 managed to reach the final goal. GMapping had similar values, with 188 reaching the first goal, and 122 completing the entire trial. ICP achieved higher performance in the second round of simulation, with 148

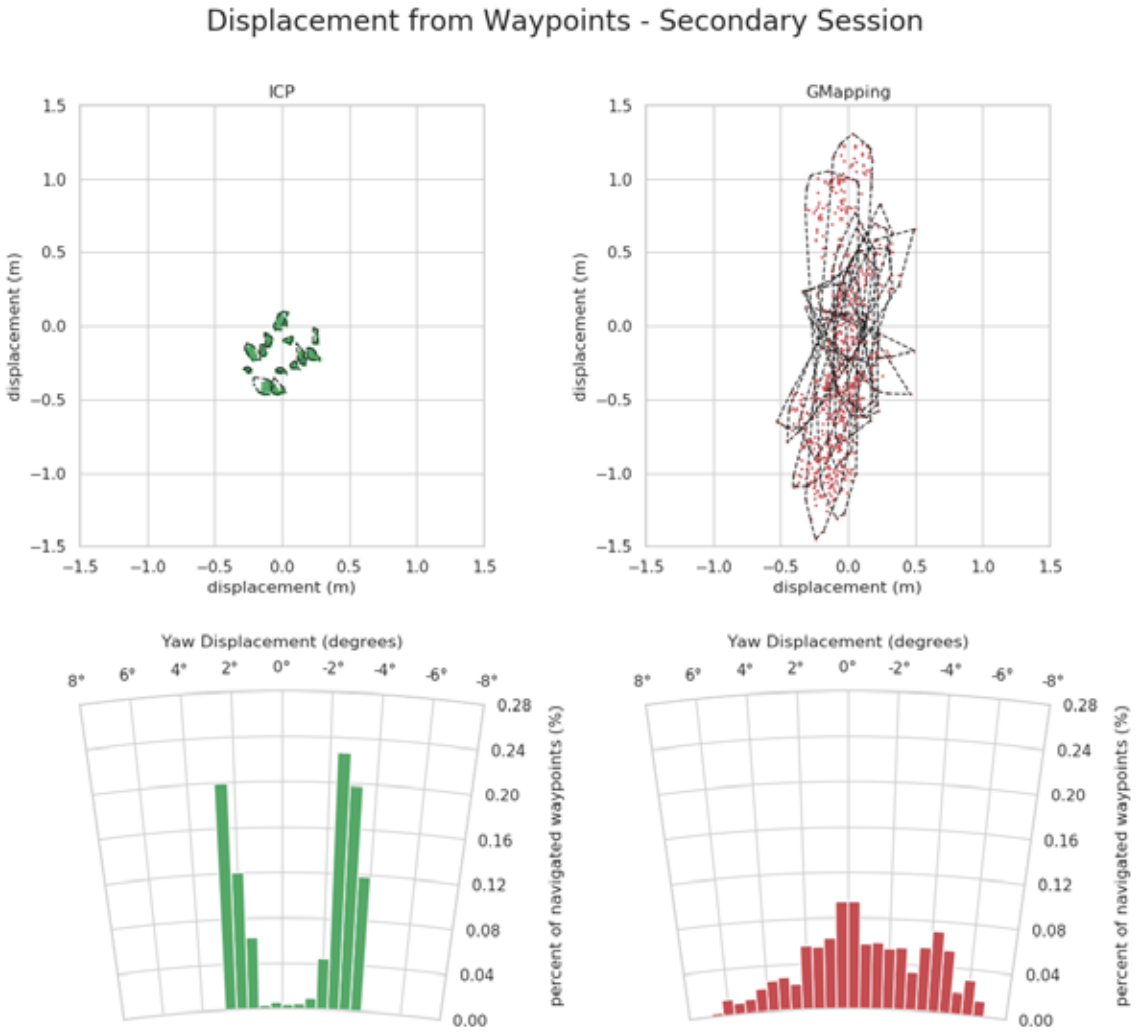



Figure 4.5: Translational and rotational displacements of Jackal UGV when navigating to waypoints in simulation, during the second navigation session. Convex hulls around data from each waypoint illustrate navigation precision within the translational displacement plots at top.

trials reaching the final goal, although only 195 reached the first goal. As expected with the additional error, GMapping had 162 trials reach the first goal, but only 14 trials managed to navigate to all the waypoints. Besides the accuracy and precision outcomes, the robustness of each algorithm demonstrates the benefits of localization using a prior map, for repeated waypoint navigation tasks in indoor environments.

### 4.3.2 Measuring Accuracy

When navigating to a waypoint, accuracy is a measure of how close the robot was to the true goal. A simple method to compute accuracy is an average of Euclidean distances:

$$\frac{1}{nm}\sum_{j=1}^{m}\sum_{i=1}^{n}||z_{ij}-z_i^*||. \tag{4.1}$$

For 2D Euclidean distance, $z_{ij}=(x_{ij},y_{ij})\in s_{\tau ij}\in\mathcal{S}_j$ and the true pose $z_i^*=(x_i^*,y_i^*)\in w_i\in\mathcal{W}$, resulting in an average radial distance. Accuracy was recorded independently of the specific waypoint measured against. For our initial session, ICP yielded an accuracy of $0.257994m$, while GMapping has a significantly smaller value at $0.105023m$. Angular orientation accuracy used the same basic equation, with $z_{ij}=q_{ij}\in s_{\tau ij}$ applied to the yaw angle, and once again $z_i^*=q_i^*\in w_i$. As seen in Fig. 4.4, both algorithms achieved similar rotational accuracy, which amounted to 0.045672 radians for ICP and 0.04284 radians for GMapping. Based on these results, GMapping has higher translational accuracy with comparable rotational accuracy, when compared to the ICP framework, using the exact origin as the robot's starting location for each trial.

When performing multiple sessions, GMapping was unable to maintain its higher translational accuracy. As seen in Fig. 4.5, the ICP framework resulted in nearly the same values as previously, with a translational accuracy of $0.255417m$ and rotational accuracy of 0.045574 radians. GMapping however became drastically worse at locating waypoints, resulting in a translational accuracy of $0.611964m$. Rotational errors seemed to be more centered with a wider range, leading to an accuracy of 0.043954 radians.

### 4.3.3 Measuring Precision

Table 4.1: Initial Session Precision

| | 2D Norm (m) | | Angular Norm (rad) | |
|---|---|---|---|---|
| Waypoint | ICP | GMap | ICP | GMap |
| 1 | **0.009754** | 0.037302 | **0.008005** | 0.010106 |
| 2 | **0.020213** | 0.043521 | **0.001827** | 0.014632 |
| 3 | **0.012137** | 0.042487 | **0.002672** | 0.014590 |
| 4 | **0.022351** | 0.086543 | **0.021018** | 0.037307 |
| 5 | **0.008230** | 0.059079 | 0.021327 | **0.009029** |
| 6 | **0.031140** | 0.089944 | **0.004168** | 0.035475 |
| 7 | **0.026128** | 0.060534 | **0.005460** | 0.007930 |
| 8 | **0.022558** | 0.090067 | **0.002955** | 0.026483 |
| 9 | **0.024736** | 0.049438 | **0.008284** | 0.036498 |
| 10 | **0.010818** | 0.048211 | **0.002223** | 0.028862 |
| 11 | **0.009416** | 0.042898 | **0.002218** | 0.013759 |
| 12 | **0.023257** | 0.047029 | **0.002852** | 0.013186 |
| 13 | **0.011861** | 0.034731 | 0.012552 | **0.008778** |
| 14 | **0.006096** | 0.035315 | **0.002980** | 0.030190 |
| 15 | **0.030237** | 0.036329 | **0.002558** | 0.026937 |

Precision can be computed using a similar averaging of the Euclidean norm, with different parameters. This time, errors were divided into subsets based on the waypoint they corresponded to, using Equation (4.2) for each waypoint $i$:

$$\frac{1}{m}\sum_{j=1}^{m}||z_{ij} - z_i^*||. \tag{4.2}$$

Within those subsets, centroids were computed to estimate how tightly packed the errors were. For translational precision, the centroids were computed as $z_i^* = (\bar{x}_i, \bar{y}_i)$, while rotational precision was able to directly use the average yaw. Navigation precision was recorded for each waypoint, as seen in Table 4.1. While there is some variability among the results, in the first session, the precision of the ICP framework is superior to GMapping at every waypoint for both translational and rotational pre-

cision, with only two angular exceptions of waypoints 5 and 13.

Table 4.2: Second Session Precision

| | 2D Norm (m) | | Angular Norm (rad) | |
|---|---|---|---|---|
| Waypoint | ICP | GMap | ICP | GMap |
| 1 | **0.011429** | 0.721628 | **0.016104** | 0.032240 |
| 2 | **0.018816** | 0.831810 | **0.001875** | 0.041254 |
| 3 | **0.012426** | 0.409511 | **0.002716** | 0.037715 |
| 4 | **0.022374** | 0.453125 | **0.024044** | 0.045671 |
| 5 | **0.008631** | 0.466506 | **0.021348** | 0.024298 |
| 6 | **0.027262** | 0.520502 | **0.005759** | 0.035139 |
| 7 | **0.025202** | 0.469325 | **0.007286** | 0.016256 |
| 8 | **0.025000** | 0.426370 | **0.005387** | 0.021191 |
| 9 | **0.026525** | 0.332063 | **0.006611** | 0.030635 |
| 10 | **0.010797** | 0.287184 | **0.002530** | 0.035593 |
| 11 | **0.010091** | 0.295293 | **0.002703** | 0.021767 |
| 12 | **0.024989** | 0.291501 | **0.002888** | 0.031338 |
| 13 | **0.012936** | 0.191098 | **0.015607** | 0.021560 |
| 14 | **0.005804** | 0.201783 | **0.002776** | 0.019334 |
| 15 | **0.031349** | 0.119085 | **0.001721** | 0.025049 |

Similar to the decline in accuracy, GMapping suffered drastically with respect to the precision realized in the translational errors of its second session. Most waypoints saw order-of-magnitude worse results, while all angular precision measurements were worse than those from the ICP algorithm, as seen in Table 4.2.

### 4.3.4 Real World Experiments

ICP localization was implemented on a Jackal UGV using a global reference map of the ABS Engineering Center at Stevens Institute of Technology seen in Fig. 4.6. A script published the same waypoints for five consecutive trials. The final waypoint was placed at the origin of the map so that the robot could return autonomously to its start location. To simulate multi-session navigation, the localization algorithm was restarted remotely before each trial began, however the hardware was not touched

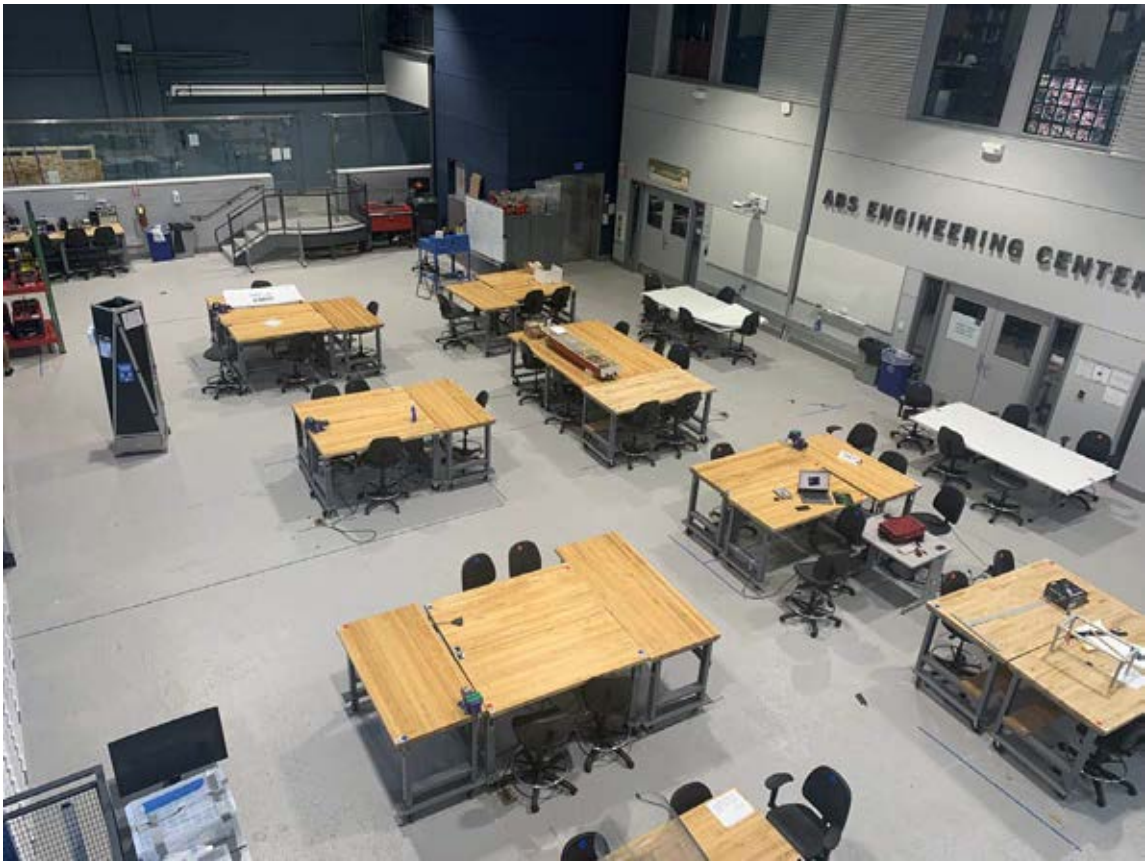



Figure 4.6: ABS Engineering Center used for real world testing of ICP localization framework on Jackal UGV.

between trials. All five trials can be seen in Fig. 4.7 where the pathways varied, but the waypoints were reached precisely.

All of the preparations to perform real world testing, and its execution, occurred on the same day. LiDAR data was gathered by manually driving the Jackal UGV around the ABS Engineering Center while recording. For complete coverage, the manual path taken was more comprehensive than our final autonomous path. The environment was small enough that our offline SLAM solution produced an accurate prior map to use in less than 10 minutes (shown in white in Fig. 4.7). After updating the localization parameters, waypoint navigation testing began. Waypoints were manually placed on the global reference map via rviz. If the waypoints were reasonable

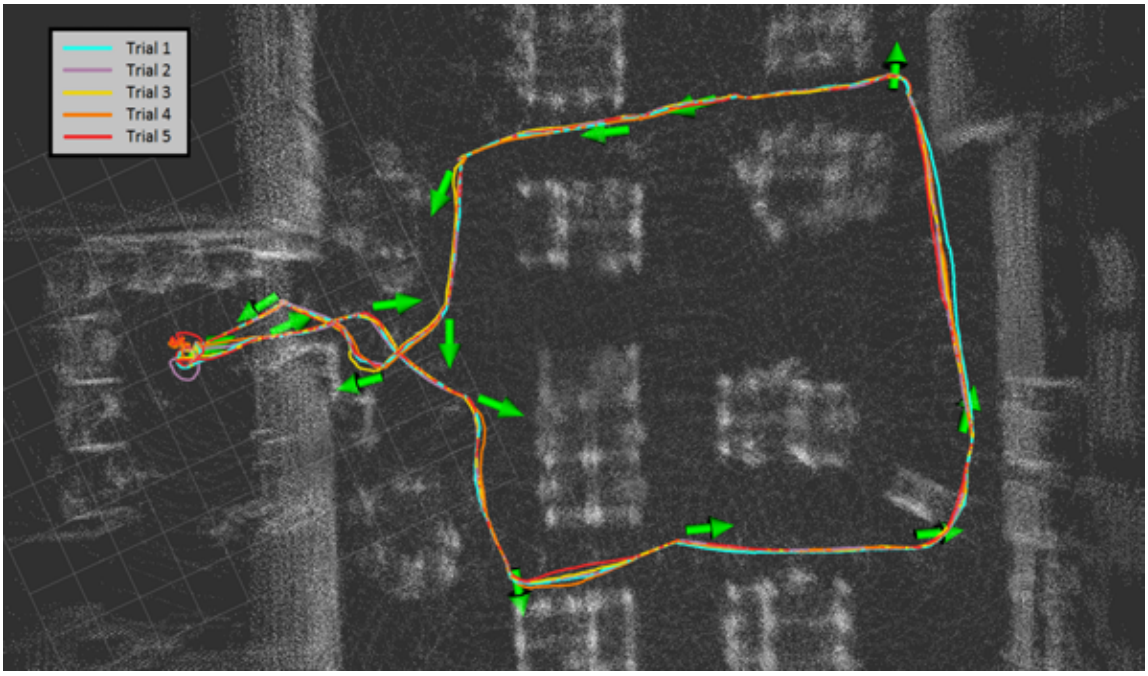



Figure 4.7: Five consecutive execution traces of autonomous waypoint navigation on the Jackal UGV using the proposed architecture of Fig. 4.2 in Stevens' ABS Engineering Center. Fifteen waypoints marked with green arrows were sent via script to run autonomously. Each new trial was started by resetting the localization software, but the hardware was not moved, to ensure true multi-session navigation. A video of this process can be viewed here:
`https://youtu.be/vSRrwgDN9kg`

(e.g., appeared to be collision-free), they were added to the autonomous navigation script. Once the script was complete, everything was put in place for autonomous navigation around the indoor environment. The total process from gathering the prior map data to defining waypoints and autonomously navigating to them took less than 4 hours.

The reference map used in this experiment included some extra data of dynamic obstacles, such as team members following the robot around. During each navigation trial, students were actively moving around this workspace as well, resulting in lidar scans that were not perfectly matched to the prior map. However, ICP localization combined with odometry achieved enough point to point matches to overcome any

mismatches due to moving obstacles, which resulted in accurate localization even with these discrepancies. Obstacle avoidance was restarted at the beginning of each trial, and only current data was used to define obstacles, which enabled the Jackal to avoid new objects that did not exist when the global reference map was created.

## 4.4 Conclusion

Using publicly available software packages, our team was able to produce a reliable and quick system for building a global reference map and performing localization that was sufficient for autonomous waypoint navigation in a previously unknown environment. The global reference map does not need to be exactly the same as the live data, thanks to the robustness of ICP combined with odometry data. This is particularly helpful in environments with constantly changing floor spaces and people actively walking around. While we were able to tackle localization for ground vehicles, the default move_base navigation package had limited success at avoiding obstacles across multiple sessions, under the accumulation of errors. We hope that the proposed architecture can serve as a foundational capability upon which the robotics community can achieve more complex task execution in GPS-denied indoor environments.

# Chapter 5

# Mobile Manipulation Platform for Autonomous Indoor Inspections in Low-Clearance Areas

## 5.1 Hardware Architecture

Figure 5.1: Reaching capabilities with small footprint.

The Jackal UGV has a surface plate with mounting holes 12cm apart to enable custom hardware integration. For stability, a new plate using all eight mounting holes was fabricated to create a higher platform which would be used as the base of the manipulator. The primary reason for this was to contain all the extra pieces of hardware underneath the plate, while a secondary benefit was to extend the range of the manipulator without compromising stability. Mounted underneath the custom plate on one side is a 19V DC voltage regulator used to power an Intel NUC. The other side has the NUC, which is used to control the manipulator and communicate with the Jackal's onboard computer. Their mounting positions and power connections can be seen in Fig. 5.2, where the plate is flipped for visibility. The batteries provided by Clearpath operate the Jackal from 29.4V to roughly 25.6V. The Kinova Gen3 arm can be powered directly by these voltages, while the NUC requires 19V

consistently. When all these electronics are powered simultaneously, a fully charged battery will last approximately 100 minutes.

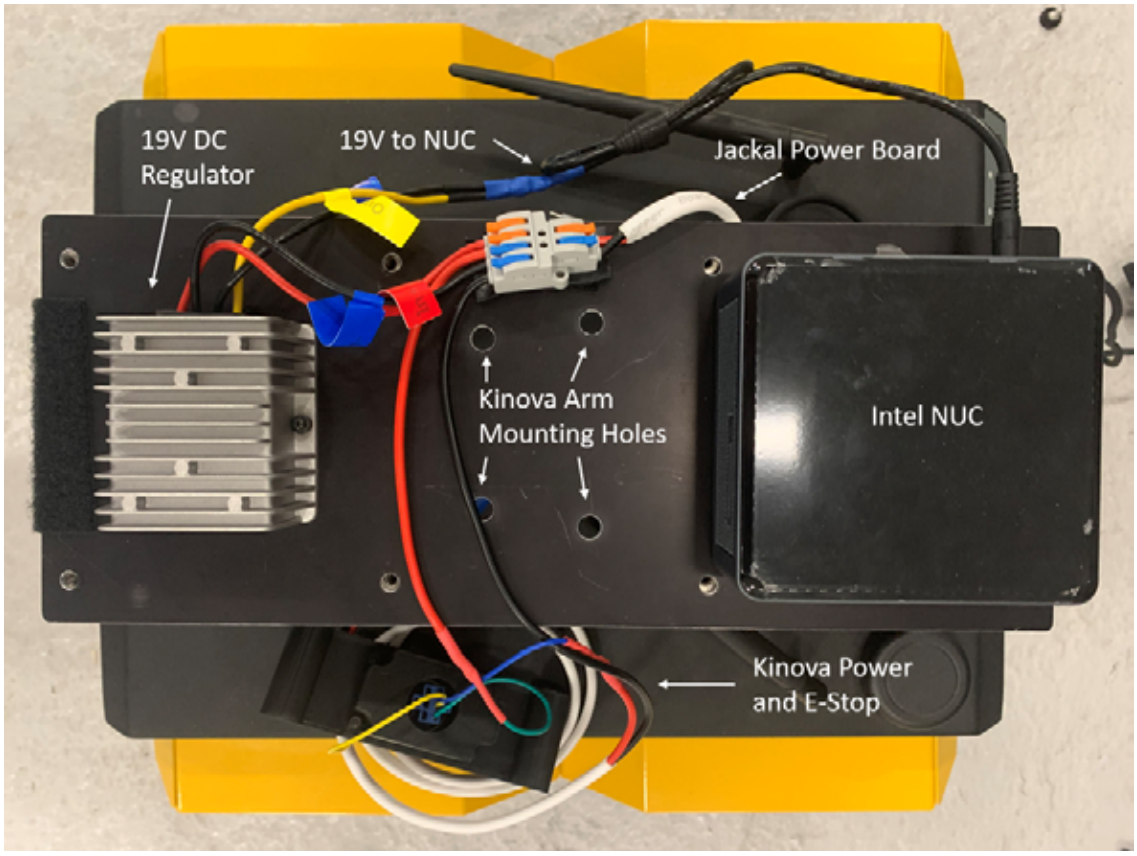



Figure 5.2: Power connections used for mounted hardware. Mounting plate is upside-down for visibility.

Data connections can be seen in Fig. 5.3 with the plate and manipulator mounted. The manipulator is centered on the Jackal, which leaves the front or rear for mounting the 16-beam Velodyne puck LiDAR. Obstacle avoidance requires no occlusion of data from the front, therefore the Velodyne was mounted towards the front of the UGV. The NUC only has a single ethernet port, which is used by the Gen3 manipulator, so a USB-C to ethernet dongle was used to connect to the onboard Jackal computer. An E-Stop switch was mounted to the front of the vehicle to cut power to the Gen3 manipulator if necessary.

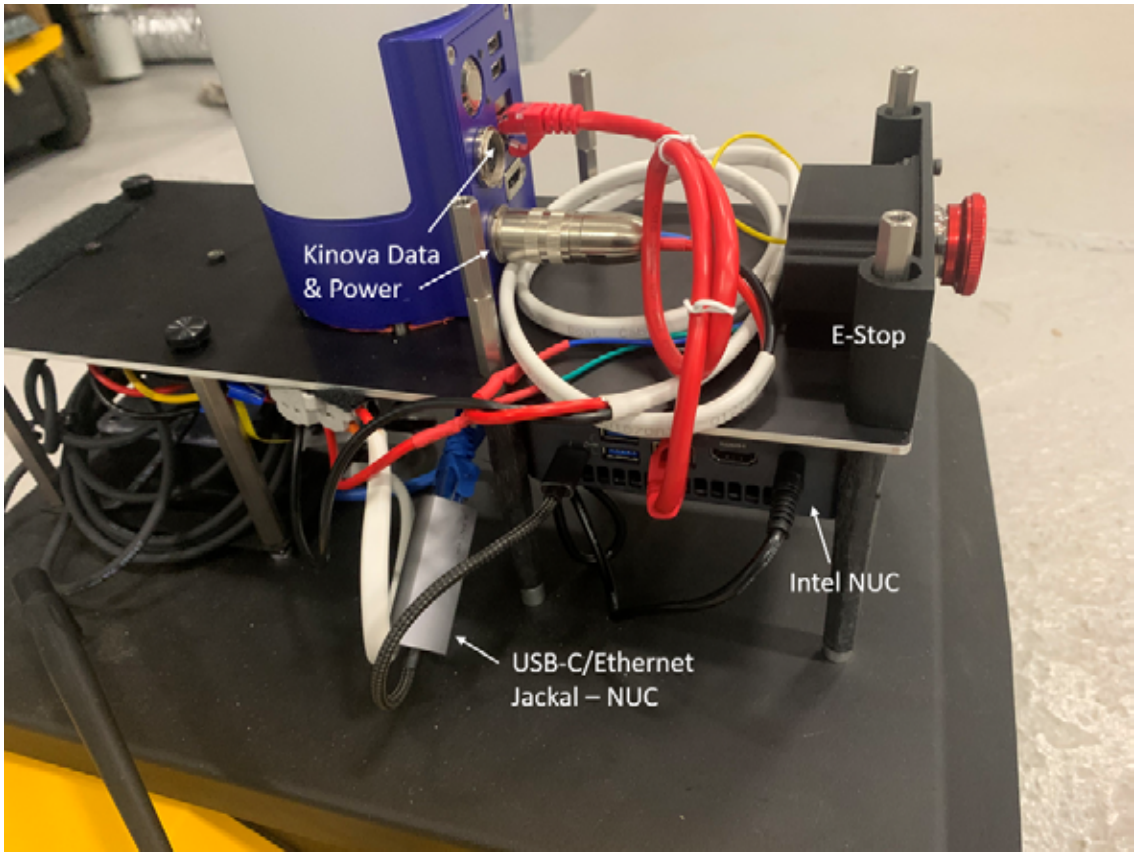



Figure 5.3: Data connections used for mounted hardware.

A diagram of the complete hardware architecture can be seen in Fig. 5.4. While the direct data connection between the Jackal's onboard computer and the NUC ensures no data loss, beginning autonomous exercises requires some form of remote activation. Therefore both computers were connected to a standalone wireless router powered by a battery that can be carried around easily in a backpack when testing. A laptop was configured to connect to the same WiFi to allow terminal control of both the Jackal and the NUC. There were some instances where the Jackal did not connect to the WiFi, however it was possible to access its computer via the NUC, reducing potential failure scenarios.

The integrated mobile manipulation platform possesses three key sensors, seen in green in Fig. 5.4. The Velodyne LiDAR is predominantly used for localization and

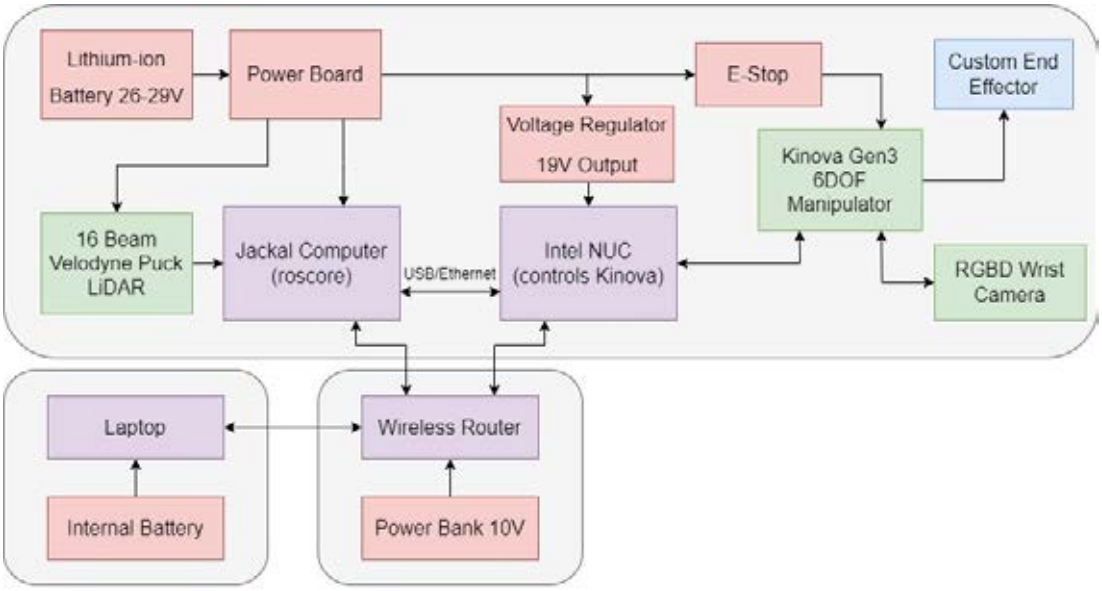



Figure 5.4: Schematic of hardware components. Grey boxes denote separate physical modules, pink describes power supplies, purple are computers, green represents sensors with feedback, while blue defines actuators.

mapping, while the RGB-D camera on the wrist of the Gen3 manipulator handles vision and range sensing tasks relevant to the arm. The manipulator also has torque and position sensors throughout, which enable precise control over its position and effort. The wrist interface can support additional sensors, but is more commonly used for actuation. In this work, no specific end effector is adopted.

## 5.2 Software Implementation

Communication between the Jackal onboard computer and the NUC was handled through the Robot Operating System (ROS). All systems have Ubuntu 20.04 installed and use the Noetic distribution of ROS, which enables users to run python3 within ROS. The wired connection on the Jackal has a default static IP that was not changed, while the NUC had to be assigned an IP address for ROS. These wired addresses were different from those used by the wireless router and seen by the laptop. However, with

the appropriate configuration, ROS was able to communicate seamlessly between all three devices with the Jackal as the only ROS master. Messages that were sent from any of the three devices were received by all three devices without packet loss until larger data sizes were attempted over WiFi. The chance of losing data over spotty WiFi was the primary reason for including a wired connection between the Jackal's onboard computer and the NUC, ensuring that locally run programs never lose data.

While the Jackal boots ROS upon startup, the NUC and Gen3 manipulator both require manual startup. The NUC itself operates as an ordinary computer with no monitor nor input mechanisms. By pressing the power button, the NUC will boot up and wait. The manipulator will operate the same way, booting up with the power button before waiting for a command. Accordingly, the laptop is crucial for initiating automation. Using the laptop to secure shell (SSH) into the NUC over WiFi allows users to start the ROS programs installed. The Jackal powerboard only releases power once the Jackal has been turned on, so the NUC can only ever be turned on afterwards. Therefore the ROS program initiated on the NUC to control the Gen3 manipulator will see the Jackal ROS master when attempting to start. Once the Jackal is on and the NUC program is running, the systems are integrated and functional.

For safety, a few software considerations were implemented. The most consequential is the "driving configuration" of the Gen3 manipulator. Seen in Fig. 5.5, the arm is lowered to reduce the center of gravity while aiming the camera to the side. This enables users to have a clear view along the side of the robot for potential interactions while maintaining stability as the Jackal navigates throughout its environment. A secondary software implementation is acceleration limits when the Jackal drives around. Using smaller acceleration limits, we were able to reduce all bouncing motion resulting from the top-heavy Gen3 manipulator. When the Jackal is

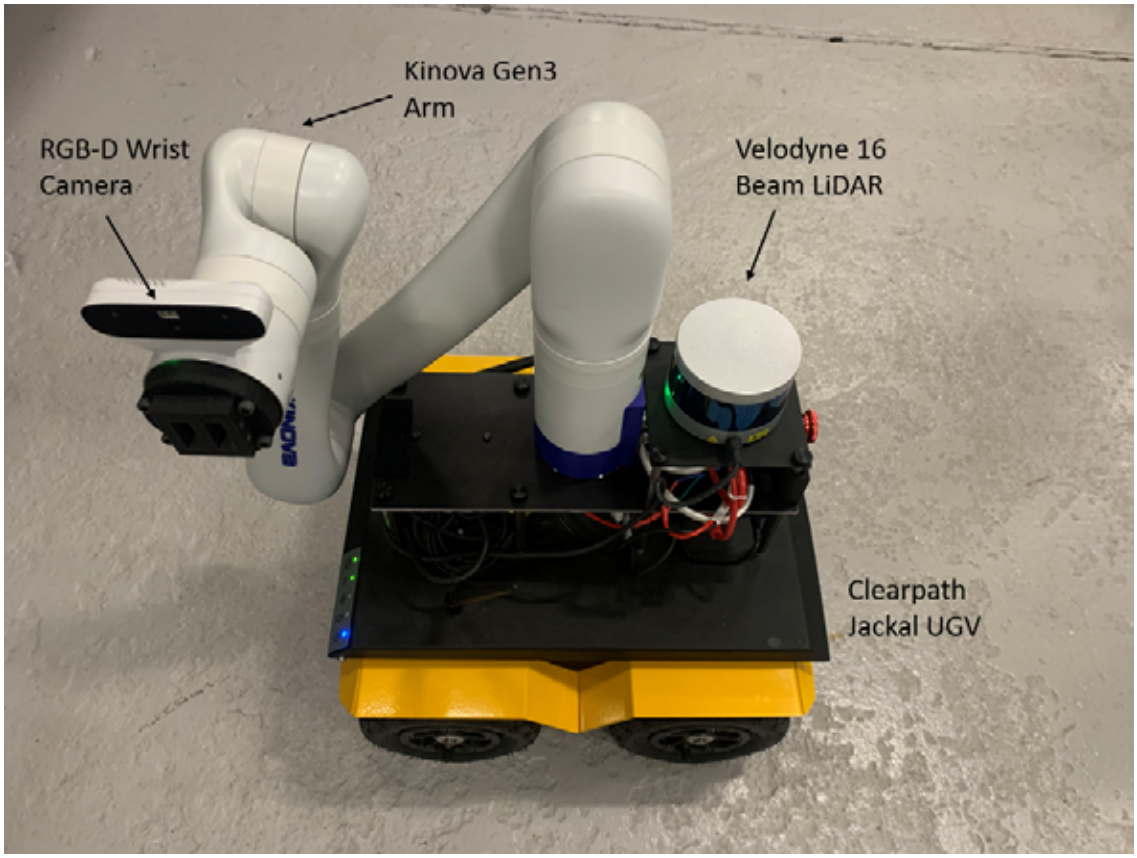
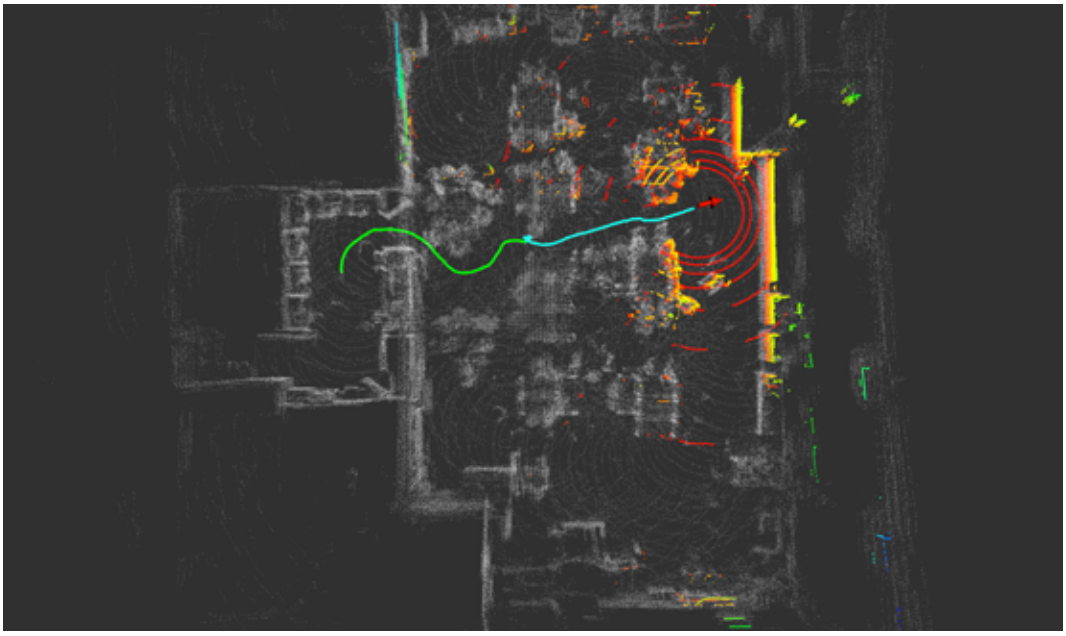
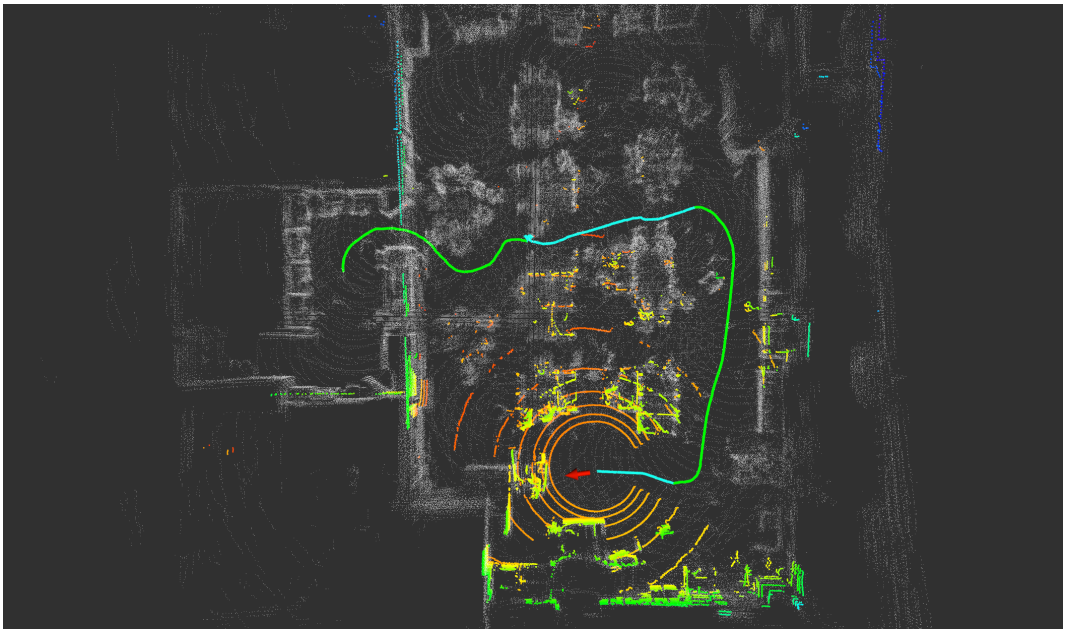



Figure 5.5: Travel configuration for safe, stable driving throughout the environment.

stationary and the arm is moving, it also has acceleration limits and boundary limits. While the $17kg$ UGV has a wide enough base to handle the full reach of the $7kg$ manipulator without tipping, extra weight mounted to the wrist could become a concern. Therefore, a software limit can be imposed upon the manipulator workspace to ensure the arm remains stable. The combination of these settings enables the mobile manipulator to freely traverse a cluttered environment with minimal vehicle footprint, while maintaining a large reach for manipulation tasks.



(a) First semi-autonomous waypoint



(b) Second semi-autonomous waypoint

Figure 5.6: Localization with Occlusion in Stevens' ABS Engineering Center. The Red Arrow indicates autonomous navigation goal. The Path in green was manually driven by remote control, while the path in blue was autonomous. The White point cloud is a global reference map, and the colorful pointcloud is the current Velodyne scan.

## 5.3 Experimental Results and Discussion

### 5.3.1 Localization with Occlusion

Two main tests were run with this hardware. The first experiment tested the platform's localization capability using the Velodyne puck LiDAR with the occlusion from the Gen3 manipulator included. Most localization performed with a LiDAR will not have such a large portion of its field of view missing. However, even with the occlusion, localization was performed accurately and reliably using our autonomous waypoint navigation package [2][1]. Localization with occlusion was tested in Stevens' ABS Engineering Center and can be seen in Fig. 5.6.

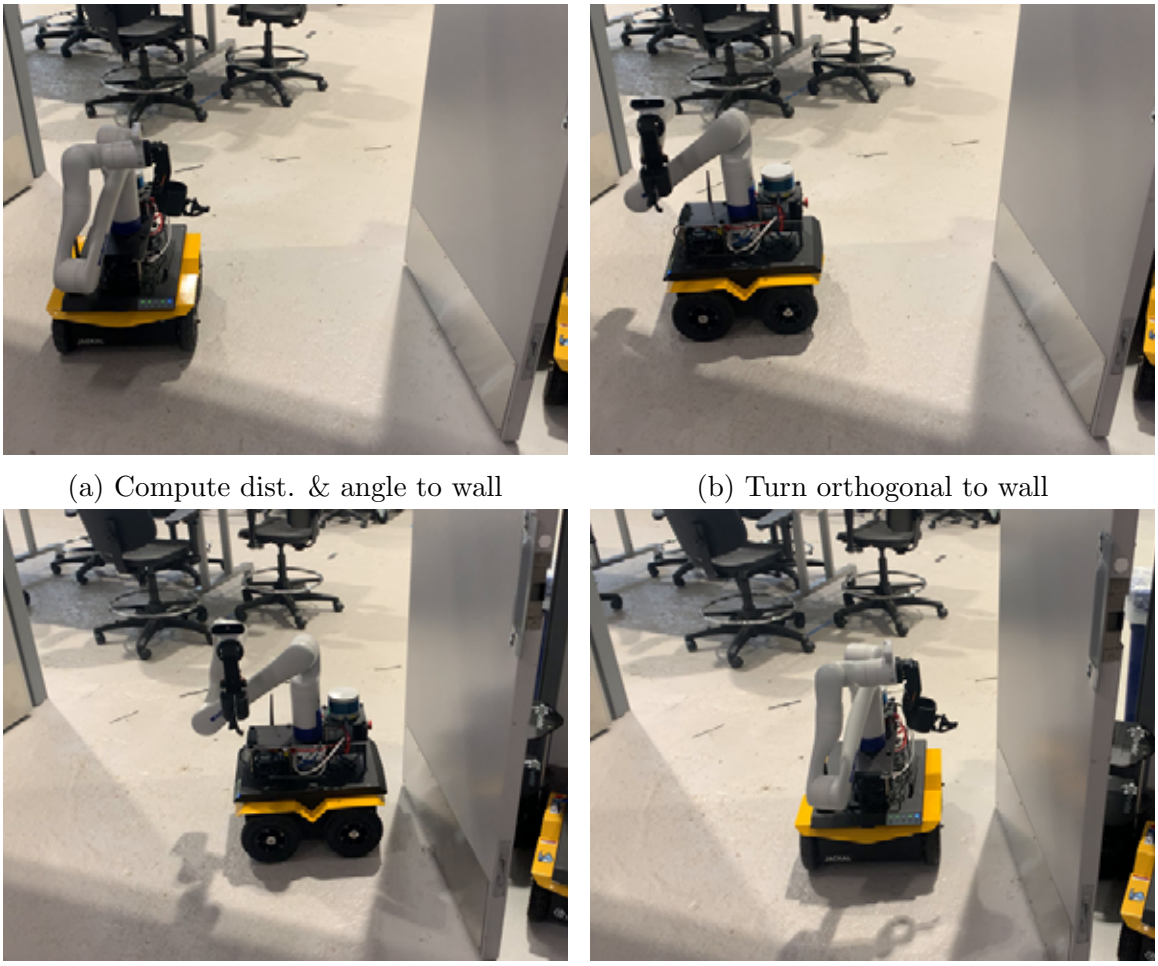
This environment was a very busy indoor atrium with upwards of 40 students working and moving throughout testing. The white point cloud data in both images was used as the global reference map for localization, and the data used to create that map was gathered approximately an hour beforehand. While testing the localization, both manual driving and autonomous driving of the mobile manipulation platform occurred. Due to the busy nature of dynamic obstacles in this environment, the autonomous driving portion occurred with fewer obstacles nearby.

Mounting the Gen3 manipulator onto the Jackal and driving around in the travel configuration resulted in a Velodyne occlusion of approximately 60 degrees, or one sixth of the complete field of view. To minimize potential failures, the occlusion was engineered to be towards the rear of the vehicle, ensuring obstacle avoidance is possible when necessary. With the occlusion limitation, localization was successfully tested with both manual driving and autonomous waypoint navigation in a busy indoor environment.

[1] https://github.com/RobustFieldAutonomyLab/waypoint_navigation

### 5.3.2 Panel Alignment

The second experiment tested the connection between the two robotic subsystems. For future manipulation tasks, aligning the Jackal and Gen3 arm to a wall panel will be required. While it may be possible to perform this task through the Jackal's Velodyne localization alone, the accuracy and precision may not be reliable enough for all scenarios. Therefore, using a combination of the arm's wrist-mounted RGB-D camera and the Jackal's Velodyne localization can produce the desired effect.



(a) Compute dist. & angle to wall

(b) Turn orthogonal to wall

(c) Drive to wall using localization

(d) Turn parallel to wall

Figure 5.7: Autonomous alignment with a panel through combined action among the Jackal and arm.

This behavior was tested on a propped open door, to show the effectiveness of alignment on partial planar structures. We assume the camera is facing some panel-like object for this process to succeed. The first step uses the depth camera data to identify both the distance and the angle between the current position of the robots and the identified panel. Once computed, the Jackal rotates towards the panel until orthogonal, using its localization to the global reference frame to ensure accurate orientation. The Jackal then proceeds to drive towards the panel, and its standoff distance is also checked using localization in the global reference frame to ensure an accurate traversal. The final step for panel alignment is for the Jackal to return to being approximately parallel to the panel to enable the maximum workspace for the Kinova manipulator to begin working as intended. This also allows the program to confirm that the robots are close enough to begin the manipulation task (if they are not, a final rotation is performed by the arm to further align with the panel). This process is summarized in Fig. 5.7. We expect this functionality to be used for placing the Jackal at specific locations in an indoor environment where manipulation tasks are required. The Gen3 arm should complete all tasks while the Jackal is stationary, before returning to the travel configuration and moving to the next location.

## 5.4 Conclusion

A custom combination of Clearpath's Jackal UGV and Kinova's Gen3 six degree-of-freedom manipulator has been proposed for indoor inspection tasks in confined areas requiring a larger manipulator reach. With a safe travel configuration and limited accelerations, the platform was stable during autonomous navigation throughout a variety of indoor environments. The hardwired data connection between robots ensures no data loss when running computationally intensive programs for solving navigation

or identification tasks. Further studies with this platform will include utilization of the image data from the RGB-D wrist mounted camera, such as detection, tracking, and visual servoing tasks. Custom end effectors will define the set of tasks that can be accomplished by this platform, which hopefully will make use of its autonomous navigation and long reach capabilities.

# Chapter 6

# Robust Autonomous Mobile Manipulation for Substation Inspection

## 6.1 Problem Description

Acoustic and TEV measurements taken from a pothead compartment's outer panel can be used to identify anomalies before they lead to equipment failure, however taking the measurements can be dangerous and monotonous. Therefore, an autonomous robotic solution has been designed around the Insight$^{TM}$ 2 handheld PD sensor from HVPD[1], which was selected to take the required measurements. The Phase Resolved Partial Discharge (PRPD) feature separates background noise from true partial discharge sources, enabling users to identify sources of concern quickly.

### 6.1.1 Partial Discharge Sensor

To use the PDS Insight$^{TM}$ 2 seen in Fig. 6.1 for repeated acoustic and TEV measurements, a program can be created. Barcode stickers as seen in Fig. 6.2 must be scanned to let the sensor know which panel is being inspected. After the barcode has been scanned by the sensor, a measurement may be collected. However, each measurement must be assigned to a barcode, thus each barcode will be scanned twice, once for the acoustic measurement and again for the TEV measurement.

To successfully scan the barcode, the sensor has a horizontal diode laser which can read a barcode when scanned between $2-5$ inches from the sticker. This assumes minimal roll, pitch and yaw differences between the sticker and the sensor. Vertical tolerance is approximately $1cm$ while horizontal tolerance varies by distance from the sticker, but can be assumed to be $1cm$ as well. These limitations require precision for

[1]https://www.hvpd.co.uk/pdsi/


Figure 6.1: The Partial Discharge Sensor (PDS) Insight$^{TM}$ 2 from HVPD, which can record Trasient Earth Voltage (TEV), Acoustic and High Frequency Current Transformer (HFCT) measurements. Using barcode stickers to identify the substation equipment of interest, measurements can be compared over time for diagnosis.

both robotic manipulation and visual barcode identification.

Acoustic measurements are intended for vented locations, where they can accurately record the sounds within a pothead compartment. Without physical contact, the sensor should be placed within $1cm$ of a vent to listen for anomalous acoustic signatures. Measurement data can be gathered with $5, 10$ or $60$ second periods. Confirmation of successful measurement is required to store data.

Transient Earth Voltage (TEV) measurements require contact with a metal

surface. For substation inspection, the panels used to cover switchgear cubicles are sufficient. Direct contact must be made with the metal surface which similarly requires minimal pitch and yaw differences, however roll does not affect the results.


Figure 6.2: An example of the PDS Insight™ 2 attempting to scan a barcode sticker.

The same time periods can be set for the TEV measurements. While it is possible for vents to be located differently on each panel in a substation, an assumption was made that all vent and TEV measurement locations are the same relative to the location of each pothead compartment's barcode sticker.

### 6.1.2 Automation Requirements

For a fully automated inspection, a robot would need to perform the functions outlined in Fig. 6.3. Each function has two primary tasks, interacting with the handheld sensor and moving the sensor around. Precise motion between barcode, vent and panel locations can be performed by a robotic manipulator. However, motion between barcode stickers needs a ground vehicle. Therefore, a mobile manipulation platform was necessary to meet the motion requirements for this automation task.

In addition to motion requirements, our accuracy and precision requirements will need specific sensors. LiDAR was used for localization when the ground vehicle is moving. Depth camera data was used to identify distance between the mobile manipulation platform and any surface expected to be worked on. Identifying the precise barcode sticker location required an RGB camera. Internal data was used to relate vent and TEV locations to the barcode sticker location for each panel. Therefore, the mobile manipulation platform needs a LiDAR and an RGB-D camera for accurate and precise task execution.

The last mechanical requirement for autonomous inspection is actuation. Buttons on the PD sensor needed to be pressed with the correct timing to scan the barcodes and record each measurement. A custom electromechanical wrist mechanism was implemented to press the necessary buttons, with signal inputs from the robotic manipulator. Power consumption can be a limiting factor, however a full autonomous inspection run is expected to take less than thirty minutes, a small frac-

tion of how long most mobile manipulation platforms can operate on a single battery charge.

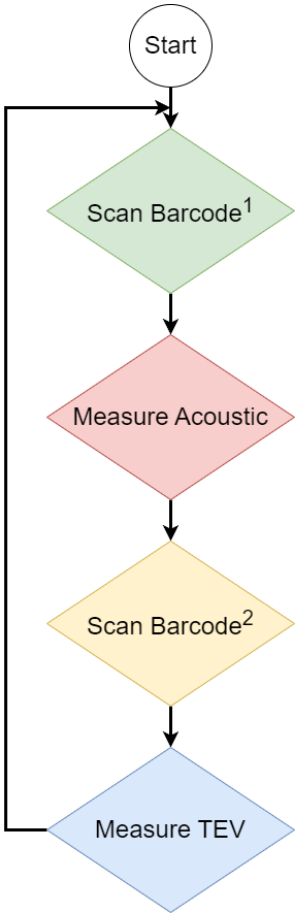



Figure 6.3: **Basic Automation Process.** For each panel being measured, the PD sensor must scan the barcode, collect an acoustic measurement, then scan the barcode a second time before measuring TEV. This process repeats until all panels have been inspected.

## 6.2 Robotic Hardware for Inspection

To meet all the requirements for autonomous inspection, we chose reliable, widely-used subsystems. Clearpath's Jackal was chosen to be the unmanned ground vehicle (UGV) integrated with a Velodyne Puck LiDAR (VLP-16). This UGV has some pre-

configured packages for LiDARs and cameras, however there is currently no package for mobile manipulation. Therefore, an independent decision was made to use the Kinova Gen3 6 Degree-of-Freedom robotic manipulator with a RGB-D wrist camera. A custom wrist mount was created to interact with the PD sensor using commands through the Kinova Gen3 arm. The integration of these three pieces of hardware satisfies all the sensor and motion requirements for the inspection task.

### 6.2.1 Jackal UGV & Velodyne Puck

The Jackal UGV was selected for its small footprint when navigating around indoor environments with narrow passageways, and for the simplicity of mounting new hardware. Larger UGVs were available with long-reaching manipulators already integrated, however they would be unable to safely navigate the typical passageways in the substations of interest. There are many options for LiDAR sensors to gather point cloud data. Prior experiments gave our team familiarity with the Velodyne Puck and the small profile of the sensor ensured minimal issues for mobile manipulation integration.

Clearpath's Jackal has an onboard computer to manage sensor processing, localization and motion planning tasks. ROS Noetic was installed to allow python3 packages to be used and ensure compatibility between the Jackal and Kinova Gen3 arm. The battery that powers the Jackal and all attached components operates between $25.5V - 29.4V$ and typically lasts for four hours on a single charge for Jackal and Velodyne operations.

### 6.2.2 Kinova Gen3 Arm & Wrist Camera

The Kinova Gen3 arm was among the manipulator options investigated, and was found to meet the requirements for this project. The minimal weight, power con-

sumption and small base plate create an ideal candidate for mobile manipulation. With a maximum reach of $891mm$, this robot manipulator is much larger than would normally safely fit on a small UGV like the Jackal. However, with precaution, safety can be assured. Kinova also offers a wrist camera with RBG-D data, fulfilling the sensor requirements for autonomous inspection.

The arm was controlled using an Intel NUC computer running Ubuntu 20.04 and ROS Noetic. The NUC requires a steady $19V$ while the Kinova Gen3 arm operates within the range of $20V - 30V$.

### 6.2.3 Wrist Mount for PD Sensor

To interact with the PD sensor, a custom wrist needed to be fabricated to push the required buttons. Using 3D printed parts and electronics, a gripper was built with the ability to push the "OK" and "$\vee$" buttons seen in Fig. 6.1. A slim profile was chosen to minimize vibrations when the manipulator moves, with a clasp to ensure the sensor does not slip from its grip. Two servo motors were integrated with rack and pinion systems with enough torque to sufficiently press the buttons as seen in Fig. 6.4.

Unfortunately, the General Purpose Input & Output (GPIO) pins from the Kinova Gen3 wrist did not operate at a high enough frequency to directly control the servos, so an intermediary was necessary. Therefore, an Arduino Nano was included with a step down voltage converter to control the button pressing behavior with inputs from the Kinova Gen3 wrist as seen in Fig. 6.5.

### 6.2.4 Hardware Integration

Mounting the Kinova Gen3 arm to a Clearpath Jackal UGV was a non-trivial task, and was described in [3]. Two main tasks needed to be accomplished for complete

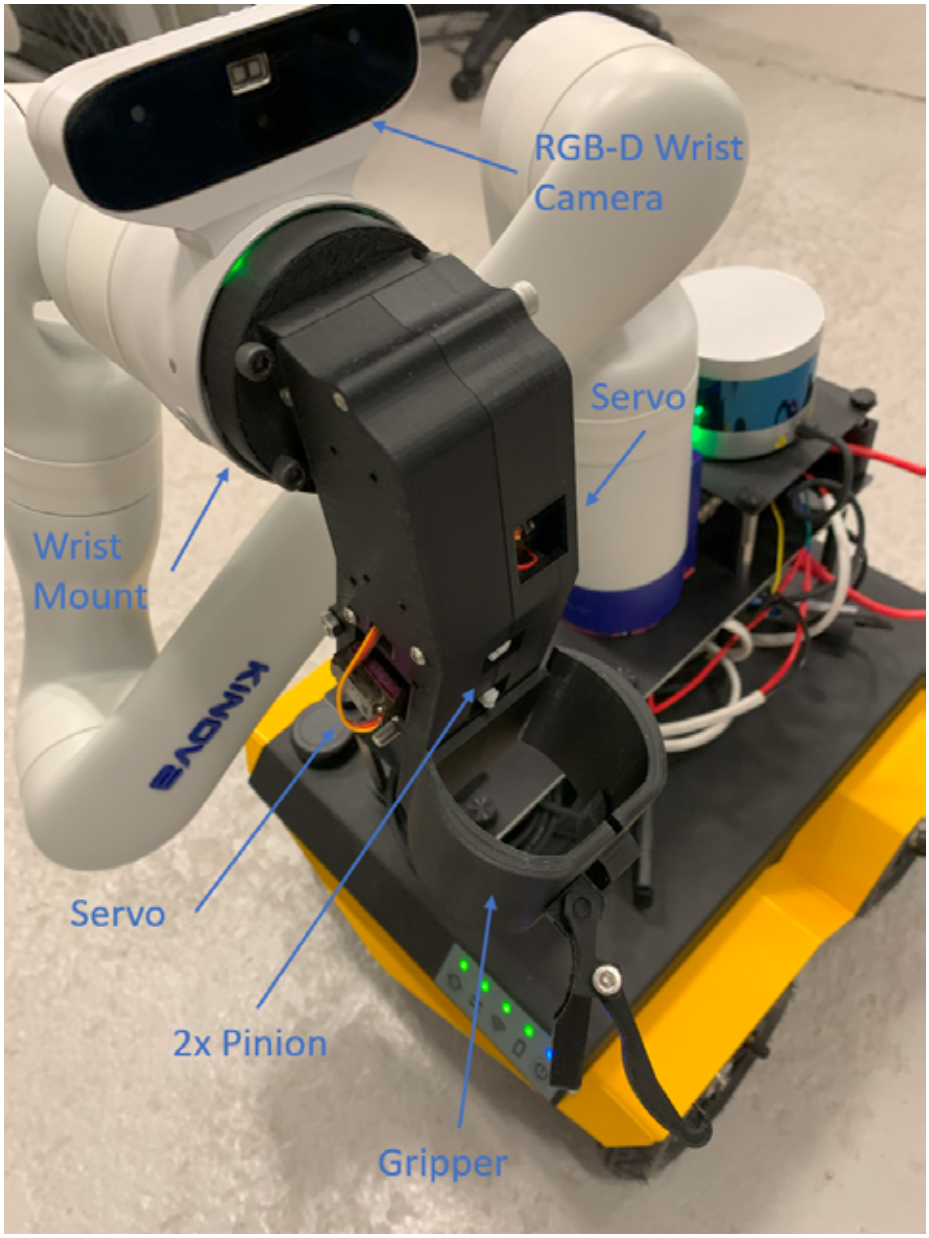



Figure 6.4: Custom 3D printed wrist to hold the PD sensor tightly, with two servos to push the necessary buttons on the sensor handle.

hardware integration. Power management was the primary task. The Jackal's on-board battery runs at $25.4V - 29.4V$ and has a power board that offers users a direct line to the battery. Those voltages meet the requirement to power the Kinova Gen3 arm, so all that was required was manufacturing a cable that fits the connectors.

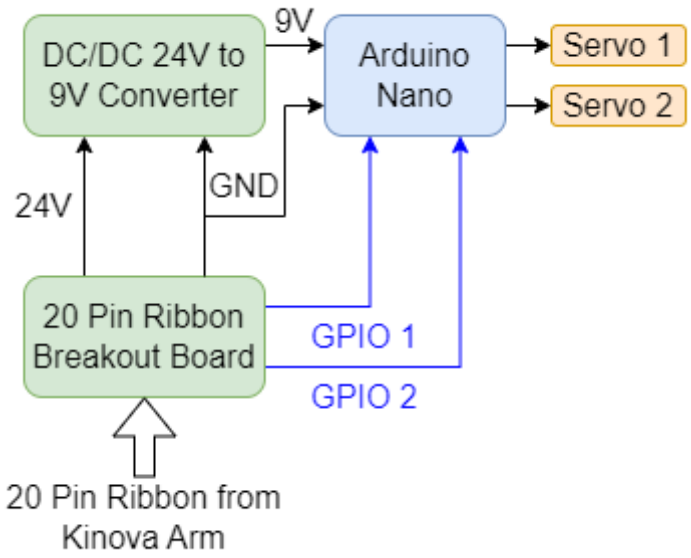



Figure 6.5: Wiring diagram of power and data connections for custom wrist implementation.

However, the Intel NUC used to control the arm requires constant $19V$. Therefore, a voltage regulator was added to the hardware. A full battery lasts approximately 90 minutes when every piece of hardware is on and operational, which is sufficient for autonomous inspections of the substations of interest.

Communication between the two robots was the secondary task. ROS Noetic offers a simple method to connect both robots for communication, with some networking. As neither the Jackal's onboard computer nor the Intel NUC have a monitor while operating as a mobile manipulation platform, an external laptop with WiFi was used for inter-connectivity and control. However, WiFi is susceptible to packet loss, especially in environments with large electrical fluctuations. Therefore, a wired connection between the Jackal's onboard computer and the Intel NUC was used for the ROS connection. The Jackal boots up as the ROS core, which allows both the power and the data connection from the NUC to rely on a separate core. Further

clarification of the hardware integration between these two robots is described in [3].

5.

Figure 6.6: Complete hardware integration in travel configuration.

Gripping the PD sensor was not included in the prior implementation of robotic integration. A custom mount was 3D printed to house the 20 pin ribbon breakout board and produce four connections in the gripper. The DC/DC converter, Arduino nano and two servos indicated in Figure 6.5 are located inside the gripper portion of

the custom-designed wrist.

## 6.3 Navigating Between Panels

One key component of repeated autonomous inspection is the ability to navigate to the same barcode sticker locations each time. Assuming that there will be a method for barcode localization via RGB camera data and precise motions from the manipulator, the ground vehicle does not require very tight tolerances for waypoint navigation. However, some checks are needed to ensure the UGV is within an operable distance from the barcode sticker.

### 6.3.1 Waypoint Navigation

A prior work describes a method to drive the UGV to the same global locations consistently with autonomous inspection in mind [2]. An example of both manual and autonomous driving with occlusion from the Kinova Gen3 arm can be seen in Fig. 6.7. While this proves that the mobile manipulation platform can drive to global waypoints successfully, there are limitations to the precision of this process. Therefore, two methods were added to the process to ensure proper positioning and alignment with respect to the desired pothead compartment panel.

The first method adjusts the UGV's orientation and standoff distance from the panel of interest. For clarity, the axes used for alignment are $Y$ and $X$, with $Y$ being normal to the panel and $X$ the horizontal motion parallel to the panel. Using the depth data from the RGB-D camera on the Kinova Gen3 manipulator's wrist, an estimate was found for the difference between the origin of the camera and the plane representing the panel. While the UGV is in motion, the Kinova Gen3 arm is folded to the traversal state, which has the wrist camera aiming directly to one side of the

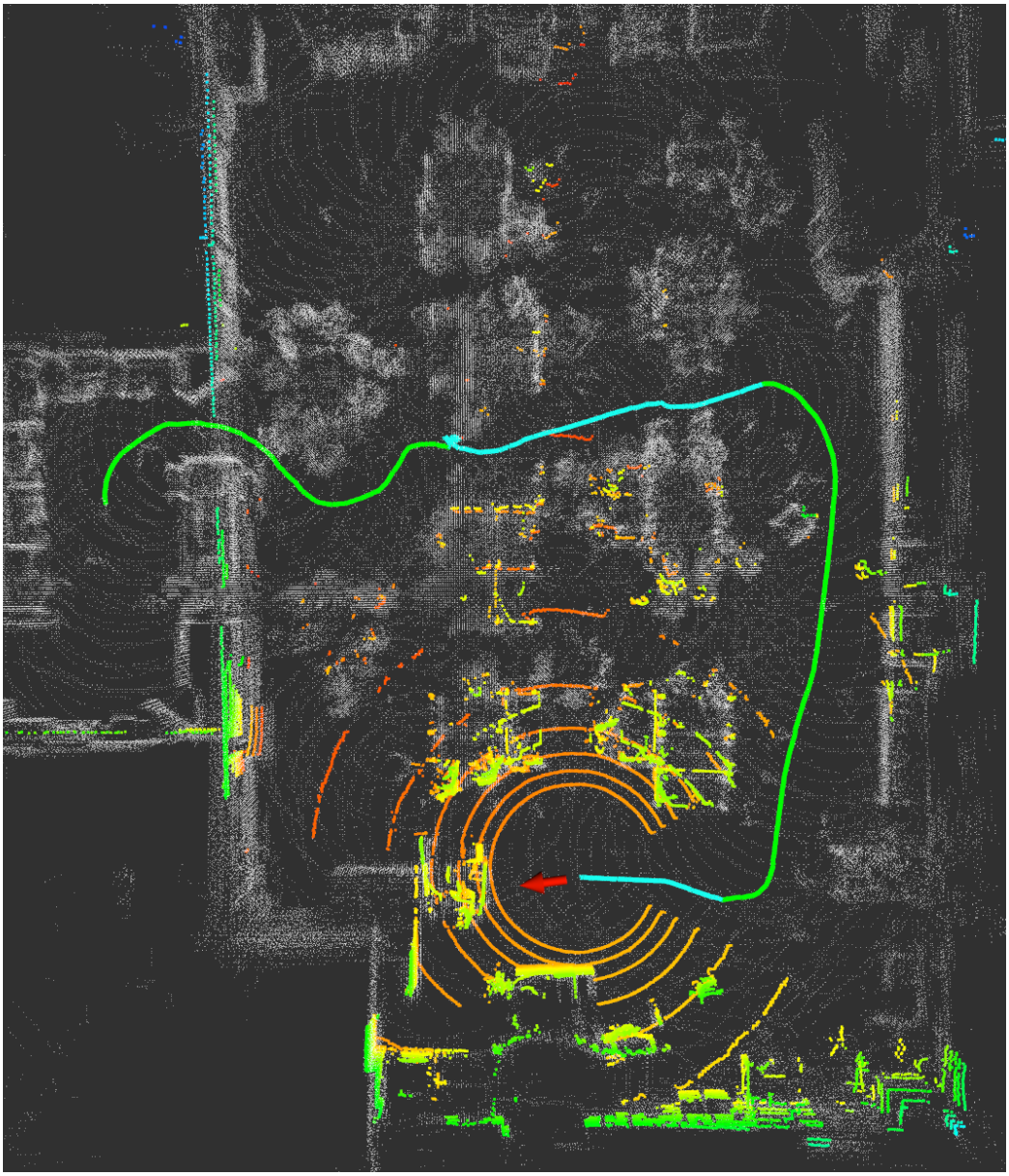


Figure 6.7: Waypoint Navigation from [3] showcasing manual (in green) and autonomous (in blue) driving of our lidar-equipped Jackal UGV with occlusions from the Kinova Gen3 arm.

Jackal. Seen in Fig. 6.8, the UGV will yaw until parallel to the panel first. If the camera is too far from the panel, the UGV will rotate to face the panel, drive forward the difference between the current position and the tolerated distance along the $Y$ axis before turning back to parallel with zero turning radius. Rotations and forward

driving use the localization data of the robot to the global reference map to ensure the commanded angles and distances are accurately captured in the UGV's motions.

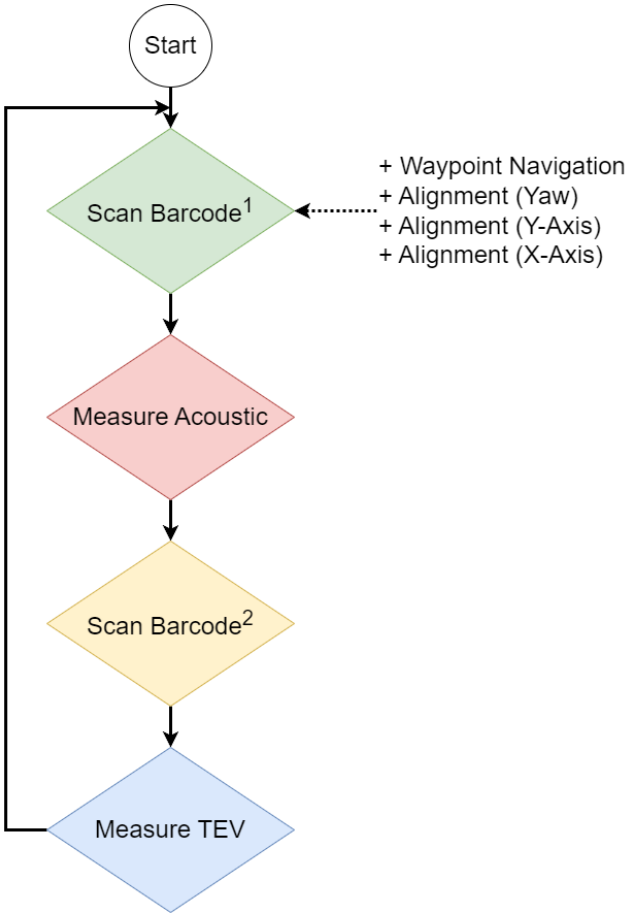



Figure 6.8: Tasks performed by the Jackal UGV for navigation and alignment added to each function as necessary.

The second alignment method occurs when the arm has begun movement to locate the barcode sticker. If the barcode sticker is located along the $X$ axis somewhere that is known to cause failures during inspection, then the UGV will need to make adjustments. Assuming the robots are already parallel to the panel, small motion directly forward or back along the UGV reference frame will suffice. Further studies of this are documented in Section 6.7, under Panel Alignment.

## 6.4 Robot Manipulator Control

Many programs exist for manipulation planning. Kinova offers examples of a driver built for complete custom motion as well as integration with MoveIt[2] in their repository[3] for the Gen3 manipulator. Alternatively, the Kinova API documentation offers a method through direct device connection[4]. Each of these methods employ two commonly used target strategies, one in the Joint space and the other in the Cartesian space.

Joint targets are easier to manage, as each joint has an encoder that measures the precise joint angle at any given moment. Given a set of target joint angles, the robot can quickly move each joint from the current location to the goal along the shortest path individually. There are cases where collisions would occur or a target angle is outside the capabilities of the manipulator. However, when operating in a small subset of the total workspace, those errors are less likely to exist.

Cartesian targets are far more complex. Inverse kinematics can be used to solve for multiple solutions, assuming no singularities. Then a pathway must be calculated between the current position and the goal position. The custom driver solution offers many methods, with the default using Rapidly exploring Random Trees (RRTs) with conservative constraints to limit the options for a successful path. MoveIt allows users to customize constraints, which is necessary as RRTs will otherwise generate new paths for each iteration of motion. The constraints supplied by the custom driver require every path to be a straight line from start to finish, and the end effector's orientation must be maintained throughout. These strong constraints work well in a small subset of the manipulator's workspace, but often result in failed paths when

[2]https://moveit.ros.org/ [62]
[3]https://github.com/Kinovarobotics/ros_kortex/tree/noetic-devel
[4]https://github.com/Kinovarobotics/kortex

the arm is extended. Due to the extended reach required for autonomous inspection, this method was insufficient. MoveIt was pursued, as custom constraints could be added as needed.

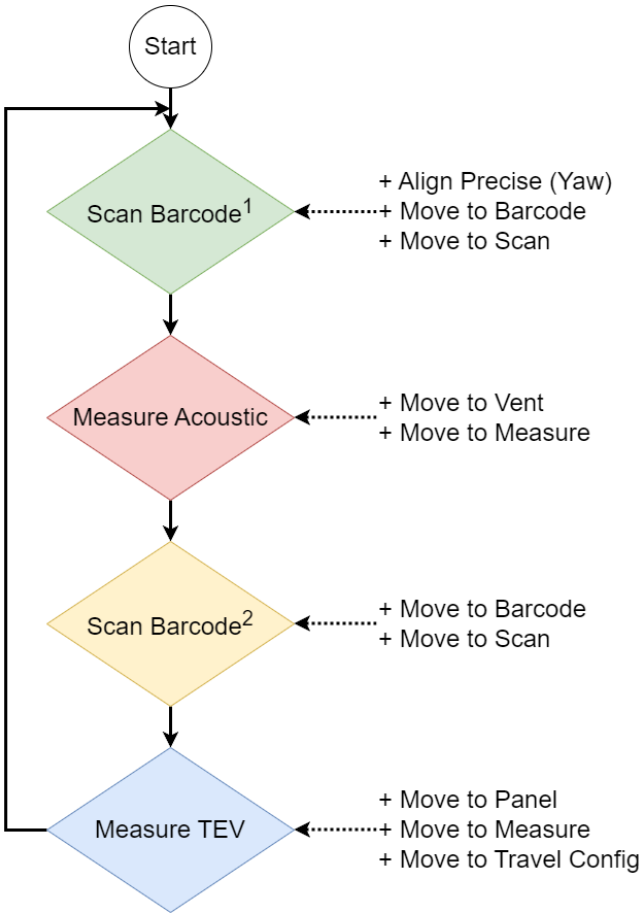



Figure 6.9: Tasks performed by the Kinova Gen3 arm for motion and alignment added to each function as necessary.

The API documented method for Cartesian motion appeared to observe shortest path constraints for trajectory planning. However, there was no guarantee of the final position, as this method would pursue a "closest pose" policy if the final end-effector location was unachievable, causing the wrist to pitch on occasion. That was not an issue when working within the bounds of the arm's workspace, so the API method was tested alongside MoveIt to identify the highest-performing method for

planning and motion control of the Kinova Gen3 arm.

## 6.5 Custom Gripper for PD Sensor

Gripping and button pressing were successfully implemented, however timing matters. Fig. 6.10 shows the required behavior to measure both acoustic and TEV data from each PD sensor function. As the manipulator moves between locations, the PD sensor must be controlled. Unfortunately, there is no feedback loop to ensure that the PD sensor is on the correct step. Therefore, the order of button presses and barcode scans is pertinent. Another feature that must be overcome is timeout, where the PD sensor will fall asleep. Certain screens have shorter timeout, but the main screen the sensor will be on while the UGV is moving between panels is three minutes.

To manage the timeout requirement with organized button pressing, multi-threading was implemented. A countdown timer runs within the second thread which ends with pressing the "OK" button and resets the timer to just under three minutes. The counting variable can be accessed from the main thread, which is used for barcode scanning as a 5-second timer is set before the arm moves towards the sticker. This allows the button to be pressed without waiting for arm motion to finish, for a higher likelihood of scanning the barcode, by mimicking human behavior for barcode scanning.

## 6.6 Barcode Identification

Safety measurements from the PD sensor require a barcode sticker to be scanned for long-term tracking of individual panels. While barcode sticker locations will be known to users and can be defined in the global reference frame, a more precise measurement will be necessary to ensure successful scanning occurs. Given that the

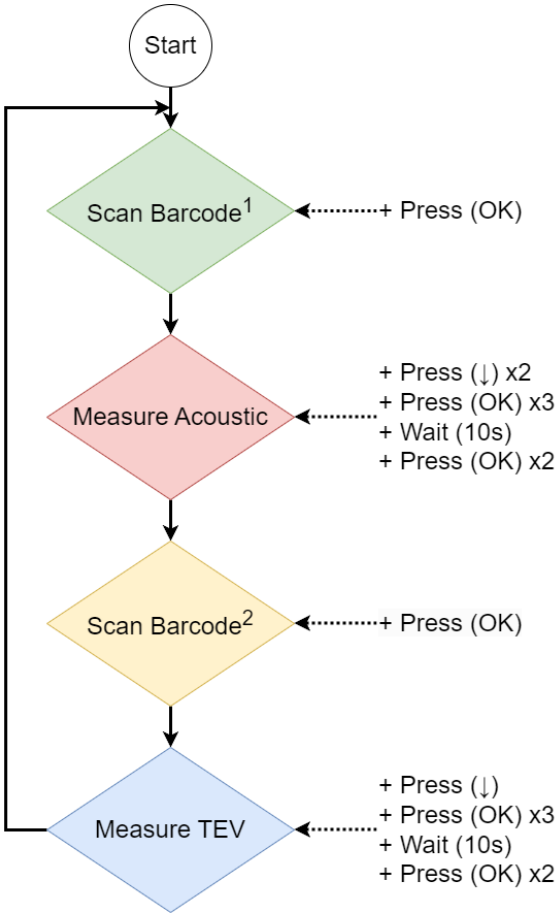



Figure 6.10: Tasks performed by the custom electromechanical gripper for PD sensor interaction, added to each function as necessary..

barcode scanning tolerance is approximately $0.5cm$ by $0.5cm$ with minimal roll, pitch and yaw, the final adjustments must be made with precision. Therefore, the Kinova Gen3 arm was used for the last step of corrections.

### 6.6.1 Camera Vision & Barcode Localization

The wrist camera was used to locate the barcode sticker precisely for manipulation tasks. RGB data from the wrist camera was published through ROS Noetic at 1080p with 30 Hz. Identifying the barcode sticker was a complex task that involved many computer vision packages and made assumptions about the environment surrounding

the sticker.

Python3 enabled the use of opencv, a computer vision package with many helpful functions. Barcode identification from RGB data is one such method, however that alone would not solve for localization accurately enough for autonomous inspection. There are objects that would be interpreted as barcodes by the computer vision function within the same view as the sticker that is necessary. Therefore, a mulit-step process was created as seen in Alg. 6.1. The first step to identify the barcode sticker is ensuring the camera is a set distance from the panel. Combined with a manual focus on the camera optimized for that distance, the best possible images will be used for identification. Then begins a for loop, with the intention of making multiple attempts in the event of failures.

Within the for loop, the image is scoured for contours and barcodes. Contours occur where pixels change quickly from light to dark and vice versa, indicating an object such as a white sticker on a gray panel. Parameters for that function have been tuned to the level of light seen in the lab testing space. Barcode identification occurs when a series of dark lines are found, which often includes nearby vents. Therefore, a method was created to identify the single contour that has at least one barcode within, assuming that must be the barcode sticker. After the contour was isolated, it was estimated to be a four sided shape, resulting in four corners. From the four corners, a center position can be calculated and using a ratio between pixel size and centimeters, a distance from the center of the camera and the center of the barcode sticker can be estimated.

For the PD sensor to scan the barcode, a vertical offset was added to the motion sent to move the arm. Pixel density measures the number of pixels per cm, a ratio to identify the location of the barcode in cm. When the camera is a set distance $d = 40cm$ from the panel, the ratio is a constant found to be $\rho \approx 65.425$.

**Algorithm 6.1:** Barcode Identification and Localization

```
moveToDistance(d)                          ▷ Set Camera distance d
focusCamera()                              ▷ Focus Camera
for Count < n do
    Count++
    C ← findContours(image)                ▷ Find "sticker"
    B ← findBarcodes(image)                ▷ Find "barcode"
    c ← checkBC( B, C):
        return c ∋ b ∈ B inside c ∈ C
    return center ← locateBarcode(c)
```

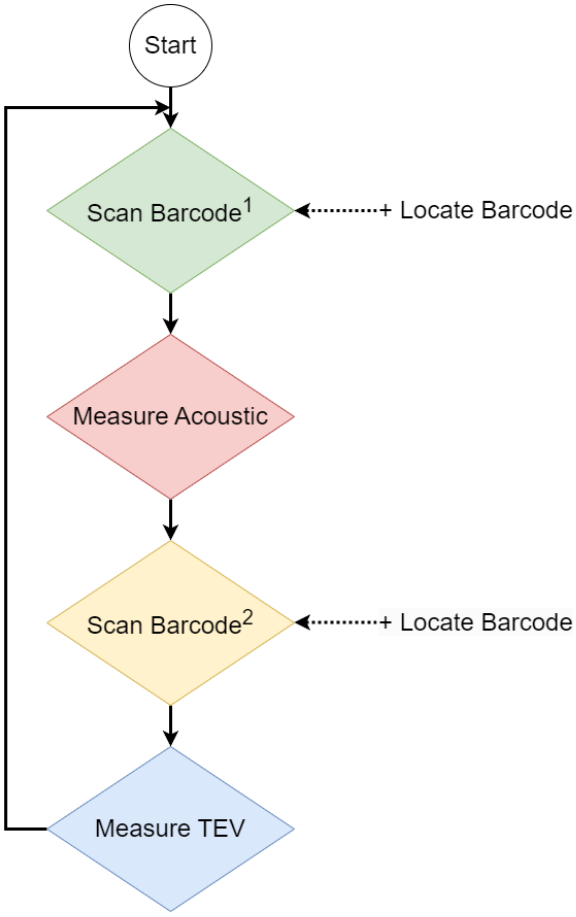



Figure 6.11: Tasks performed by the wrist camera for $X$ and $Z$ axis localization added to each function as necessary.

A full example of barcode localization can be seen in Fig. 6.12, including estimated distance between the sticker and camera FOV centers' in $cm$. Eq. 6.1 shows for measurement $i$, the calculated distance from center to center is $C_i$, where $x_p^c$ indicates the $x$ dimension of the camera's calculation $c$, by number of pixels $p$. After multiplying by the pixel density $\rho$, the value is transformed to $cm$, resulting in $x_i^c$, the value used for arm manipulation necessary for tasks seen in Fig. 6.11.

$$C_i = \left(x_p^c, y_p^c\right) \rho = (x_i^c, y_i^c) \tag{6.1}$$

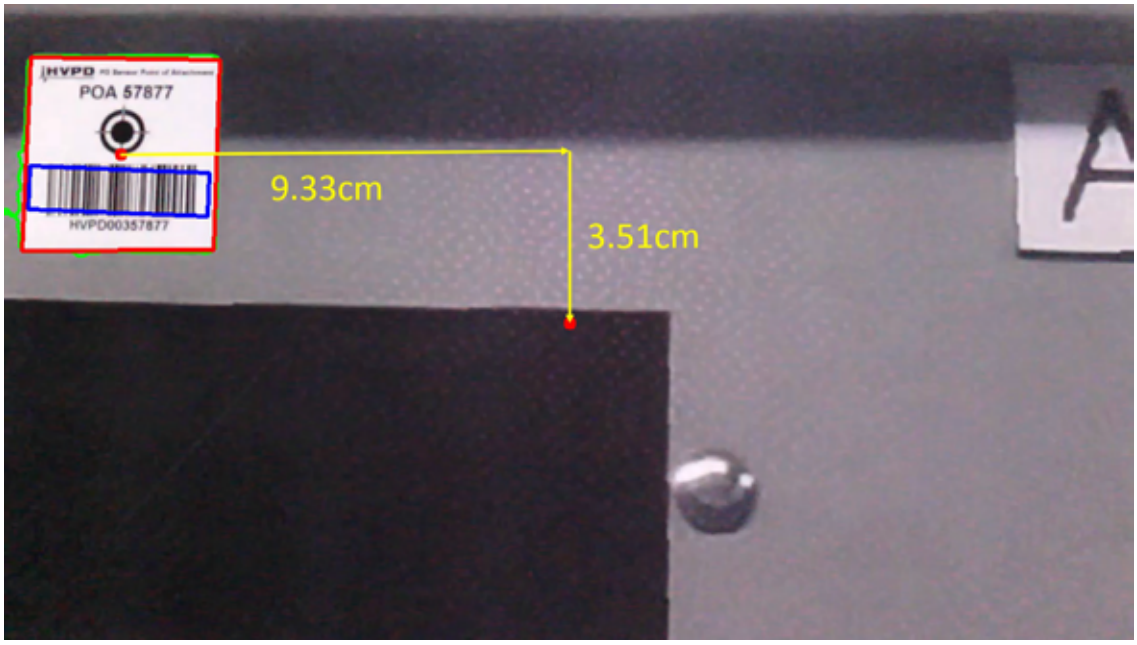



Figure 6.12: Example of barcode localization. Blue box identifies barcode, green contour finds rough shape of sticker, red box is a four sided approximation of the green contour. Red circles indicate center of camera FOV and calculated center of sticker. Distance estimates included.

## 6.7 Experimental Testing

Quantifying success of each moving part is difficult when the tolerances vary drastically and there could be many underlying issues resulting in a single failure. Therefore, data was gathered for subsets of each system that are relied on to make decisions. Waypoint navigation [2] tested the precision of the ground vehicle to approach locations in 2D space. Further studies on the panel alignment and manipulation tasks are provided here.

### 6.7.1 Barcode Localization

Described in section 6.6 via Alg. 6.1, the task of identifying and locating the barcode sticker has many steps. Ablation testing was done to test the effectiveness of each step towards the final goal of a successful pose estimation. While it is possible to take measurements from the same location each time, the results would lack the diversity needed to ensure success within the expected workspace. However, ground truth is difficult to model effectively for varying poses. One method is to calculate kinematics for the arms true location. That would require the arm to be mounted a known constant distance from the barcode sticker and maintain that relative transform throughout testing. However, a simpler method was implemented which tests both the barcode localization and the motion controller simultaneously.

### 6.7.2 Cartesian Space Motion

Cartesian motion controllers were tested alongside some barcode localization algorithms in a unique pose estimation problem. The barcode sticker was placed in a static location on a panel as the arm was mounted to a desk. A custom joint space location was implemented as a starting configuration for the arm which was the bot-

tom right corner of the camera's workspace seen in Fig. 6.13. Random numbers were generated for the horizontal and vertical motion of the arm within the camera's workspace of $20cm \times 10cm$ using cartesian motion planning for multiple controllers. The exact values in $cm$ that the arm was supposed to move were recorded during each test as $M_i = (x_i^m, y_i^m)$. Once motion had completed, the barcode localization process calculated $C_i$ from Eq. 6.1. Given the static locations of the center of the sticker and the starting position of the arm, we can use $C_i + M_i$ to estimate the difference between the two static locations.

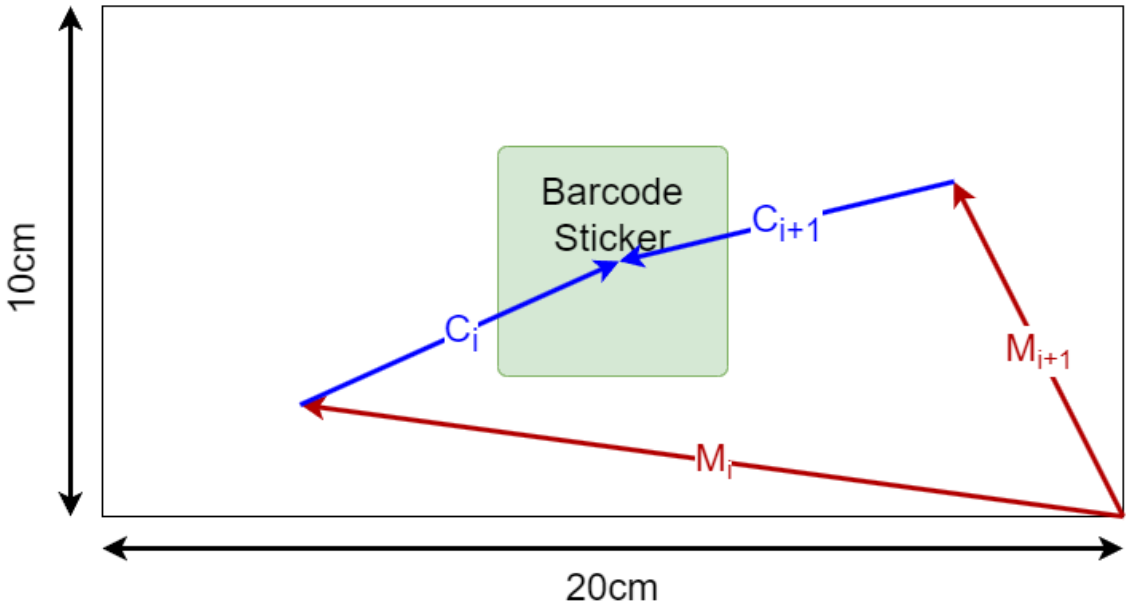



Figure 6.13: Example showing method for testing Barcode Identification with Cartesian Motion simultaneously. The arm starts in the bottom right corner before moving a known random distance within the workspace $M_i$. The camera is then used to calculate the distance $C_i$ to the center of the barcode sticker. Examples $i$ and $i+1$ are shown.

Three main options for motion control were given code examples for the Kinova Gen3 arm. All three have no issue with Joint space control, moving from the current joint angle positions to a set of goal joint angles. Each method also accepts constraints, such as objects occupying part of the arm's workspace, to plan a collision

free path. However, Cartesian space motion and control varied between the different methods.

The driver built for the Kinova Gen3 arm offers a highly constrained controller for cartesian motion. All linear motion by the end effector must result in a direct line path and the end effector must maintain the same orientation throughout the trajectory. While there are use cases that need such a tight set of constraints, the operational workspace is drastically reduced. Given that limitation, the built in driver did not meet the requirements for this project.

On the opposite side of the spectrum, MoveIt offers a completely unconstrained cartesian controller. Trajectories are planned using rapidly exploring random trees (RRTs) which results in varying paths even from the same starting and ending poses. To combat this behavior, custom constraints can be added to the motion planner manually, such as maintaining end effector orientation. For the purpose of this project, two main constraints were considered for cartesian path planning. The first constraint is used for direct forward and back motion, where the constraint is a rectangular prism in the $Y$ direction. Extended quite far forward and back, the constraint limits horizontal and vertical motion to a $5cm$ bound. This motion enables the arm to move directly towards and away from the panel as needed. The other constraint is essentially the opposite, where forward and back motion is bounded to $5cm$ while the horizontal and vertical workspace is extended out. With this constraint, the arm will be able to move parallel to the panel.

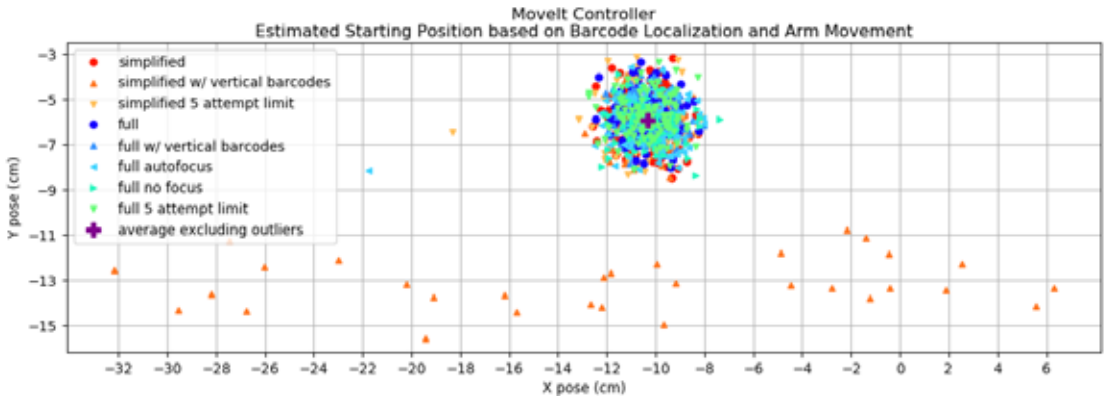
Finally, the API documentation offers a method to send joint and cartesian goal poses directly to the arm for internal planning. While there are some constraints, cartesian motion planning was successful even in the far reaches of the workspace, making this method feasible for the requirements of this project. Therefore, both the MoveIt and API methods were tested with the barcode localization methods to find

the ideal motion planner.

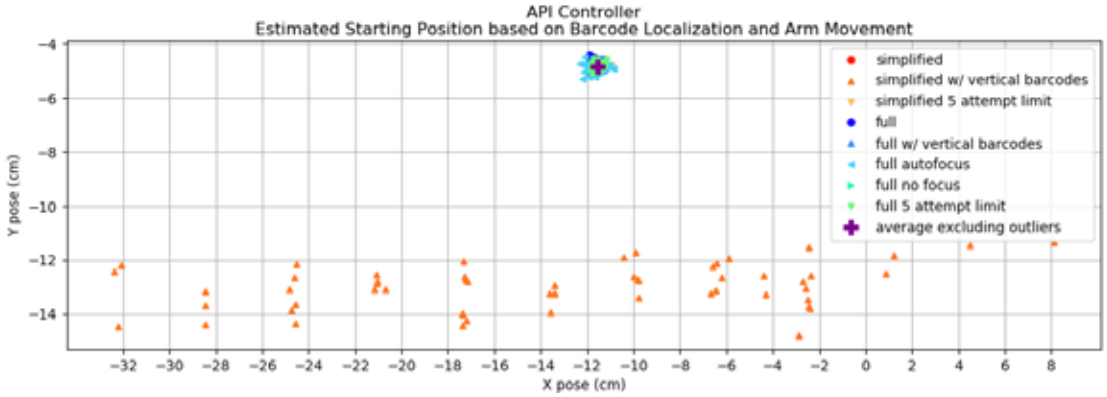
Trials of eight different computer vision methods were tested 100 times each. Two primary methods were used, the "full" method is as described in Alg. 6.1 and searches for a contour with a barcode inside to identify the sticker and therefore the center of the barcode sticker. The alternative method labeled "simplified" only considers the barcode itself by skipping lines $5, 7-8$ in Alg. 6.1, and knows that the barcode exists in the lower half of the sticker but is otherwise centered. Both methods have functions to remove any vertical barcodes, a 50 attempt limit before giving up and a manual focus based on set distance from the panel. For ablative testing, the vertical barcode limitation was removed, and the limit was dropped to 5 attempts. Changing camera focus was tested only on the "full" method and the results are independent of the rest of the localization process.

After gathering all eight trials worth of data using both the MoveIt and API motion controllers, Fig. 6.14 shows the estimated starting position of the arm relative to the barcode sticker. True positive results are clustered together, where small errors such as motion error or calculation error appear. The total range of this error is important for inspection tasks as the PD sensor requires a tight precision to scan the barcode successfully. Outlier data indicates false positive data, which comes exclusively from errors with the barcode localization method using camera data. Between both motion controllers, the simplified method without the removal of vertical barcodes clearly results in the most false positives.

An average of the true positive cluster is added to Fig. 6.14 for an expected ground truth. Using the average, we removed the false positive values to get a clearer understanding of the true positive errors seen in Fig. 6.15. The scale between the two parts were maintained to show the quantitative difference between the motion controllers. While MoveIt offers freedom of choice with regards to constraints, the



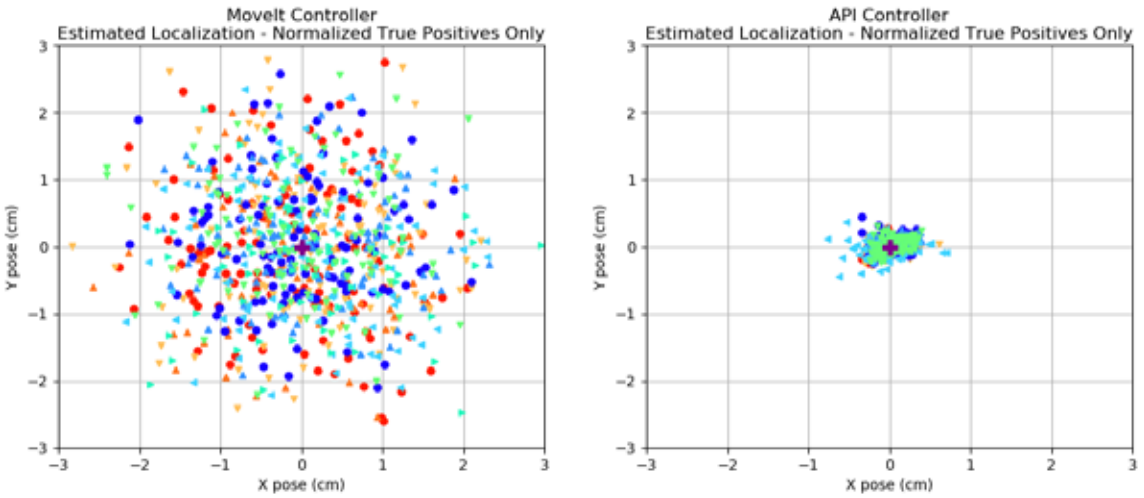

(a) Pose estimation of the arm's starting location relative to the barcode sticker using the MoveIt controller.




(b) Pose estimation of the arm's starting location relative to the barcode sticker using the API controller.

Figure 6.14: Estimated Starting Pose using $C_i$+$M_i$ for calculations. Both true positive and false positive results are shown. Two motion controllers are tested against the same barcode localization methods. 100 tests were performed per barcode localization method. An average of the true positive cluster is included for error analysis.

API motion controller is vastly more precise. A close up of API controller true positive results visualizes the calculated bounds $[-0.522cm, 0.522cm]$ in the horizontal direction and $[-0.330cm, 0.330cm]$ in the vertical direction using $3\sigma$ for 99.7% confidence of a normal distribution. The extra variance in the horizontal direction could be caused by calculation errors from the barocode localization method, as the bound is smaller if we exclude the "full autofocus" method. Compare those ranges to the $[-2.828cm, 2.828cm]$ and $[-3.004cm, 3.004cm]$ bounds found using the MoveIt con-




(a) Using the MoveIt controller, the standard deviations were calculated to be $\sigma_x^{mi} = 0.943$ and $\sigma_y^{mi} = 1.001$ which results in ranges of motion $[-2.828cm, 2.828cm]$ for horizontal and $[-3.004cm, 3.004cm]$ for vertical, assuming normal distribution.

(b) Using the same scale as the MoveIt controller figure, it is clear the API controller is significantly more precise for motion. Standard deviations of $\sigma_x^{api} = 0.174$ and $\sigma_y^{api} = 0.110$ results in $[-0.522cm, 0.522cm]$ range for horizontal motion and $[-0.330cm, 0.330cm]$ range for vertical motion.

Figure 6.15: True positives are expanded for a closer look.

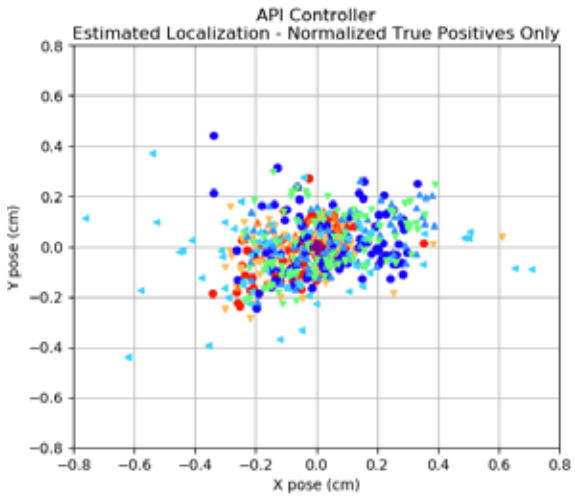
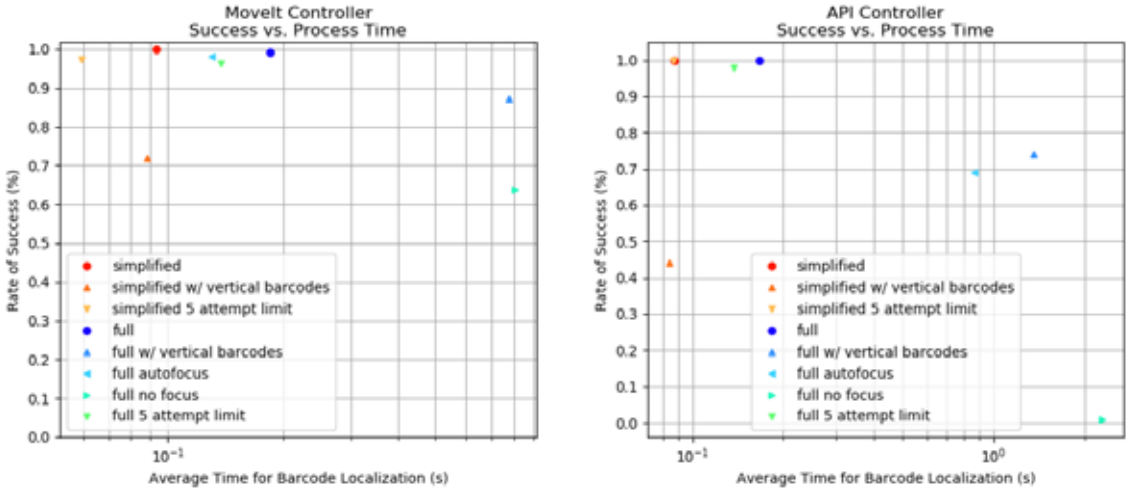



Figure 6.16: Closer look of normalized true positive results with the API controller.

troller, and it is clear that the API controller will be necessary for precision.

However, the experimental results were created to decide on not only the mo-

tion controller, but also the barcode localization method. Fig. 6.17 shows success rates and processing time of each method for both motion controllers. Success rate was calculated as the number of true positive results divided by the sum of the true positive, false positive and true negative results. There were no false negative results as being unable to find the barcode is a true negative. Similar success rates were seen between the two motion controllers, indicating independence of success rates. The only exceptions to this are the no focus and auto-focus methods, which were tested on the API version after a fresh bootup while the MoveIt data was collected after a prior data gathering session.




(a) MoveIt controller results for successful barcode localizations.

(b) API controller results for successful barcode localizations.

Figure 6.17: Two primary camera methods were chosen for base comparisons, "full" and "simplified". Ablative testing was done from each baseline, such as removing the limitation on vertical barcodes, using an autofocus or no focus at all, or reducing the number of attempts before ending the search. Time is represented on a log scale, while success rate goes up to 1.0 for 100%. Comparable success rates were seen as with each motion controller.

The "full" and "simplified" barcode localization methods out performed the rest, both operating at 100% over 100 trials with each motion controller. While the 5 attempt limit of simplified also hit 100% with the API controller, it did not with

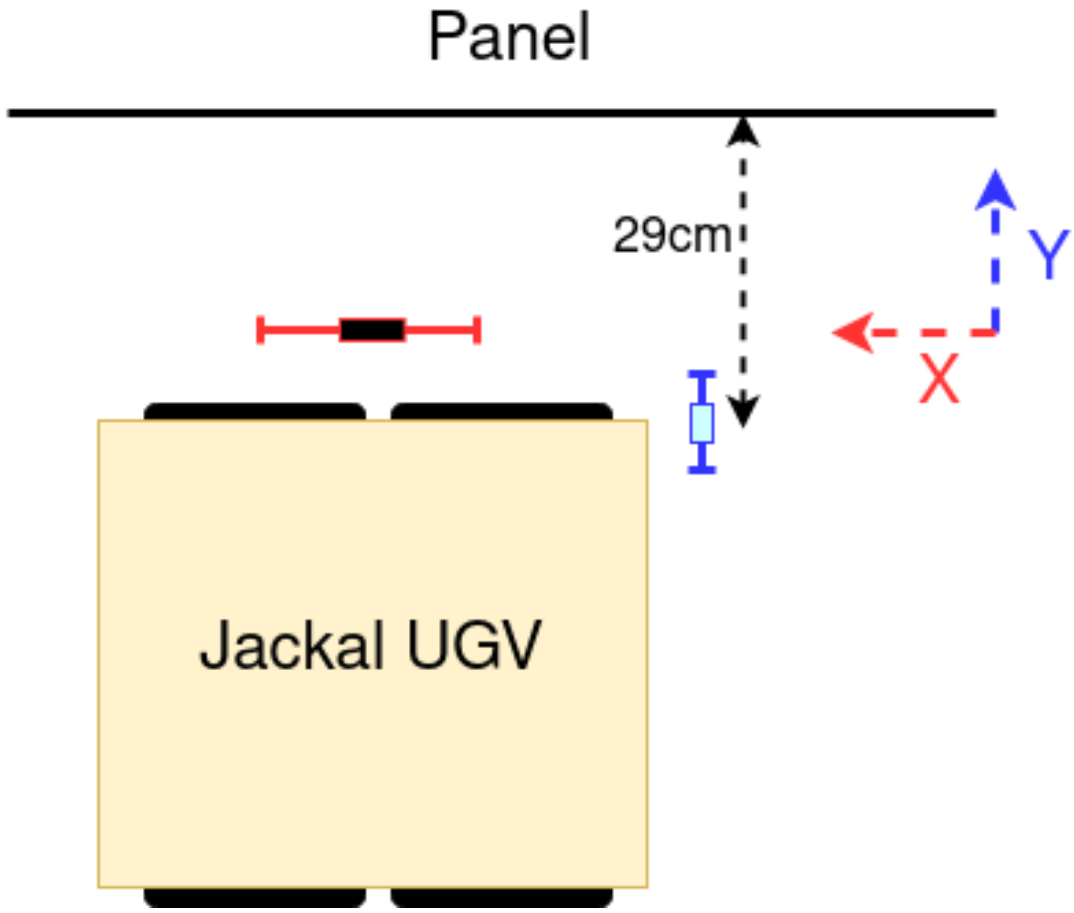



Figure 6.18: Top down visualization to scale of $X$ and $Y$ axes alignment between UGV and a panel. $Y$ axis shows full range and standard deviation in blue. $X$ axis shows full range of camera view with blackout centered section.

the MoveIt controller, indicating some possible failure modes. Process time was also considered to choose the best algorithm for barcode localization method. In both motion controllers, the simplified method reduced processing time significantly over the full method, therefore the simplified method was chosen for the final experiment.

### 6.7.3 Panel Alignment

The last functional requirement for autonomous inspection is the UGV placement for the arm to inspect successfully. Two methods were introduced for alignment, one

which attempts to correct the yaw and $Y$ axis distance to the panel and another which aims to handle cases where the barcode sticker is unable to be measured from the current position via the $X$ axis.

Using the wrist camera's depth data, a planar estimation identifies the difference between the UGVs orientation and distance to the panel. By making this method closed loop using the UGVs localization for rotation and translation measurements, accurate positioning can be acquired. Trials were run where the UGV was given a waypoint for inspection where angle and distance corrections occurred. The original waypoint was approximately $1.6m$ from the panel, with a given goal of $30cm$ and less than 5 degrees error from parallel. Testing with 100 trials showed a mean of $29.3cm$ and standard deviation of $1.73cm$, shown in Fig. 5.7. The arm inspection process was tested from $24cm$ to $33cm$ distances every $1cm$ to ensure successful measurements. Although the goal was $30cm$, the Kinova Gen3 arm is capable of performing inspections throughout the full range of $[24cm, 33cm]$ values in the $Y$ direction.

While the $Y$ direction can be managed through depth data and global localization, the $X$ direction has a strict limit. To identify the barcode sticker, the camera is moved to $40cm$ from the panel, which gives a viewing workspace of $20cm \times 10cm$ to the camera. Therefore, the range of error for being able to view the barcode sticker is $[-10cm, 10cm]$ in the $X$ direction, as seen in Fig. 5.7. Although the barcode sticker is quite high on the panel, even at the edges of that range, the arm can manage all the inspection tasks. Curiously, failures were found to exist within that range. In particular, the center of the range results in failures, indicated by the black box in Fig. 5.7. Approximately $[-3cm, 3cm]$ range can fail, therefore an extra alignment method was added. If $|x_i^c| < 3cm$, the arm is sent back to travel configuration and the UGV travels forward along the $X$ direction until the barcode search position will be $5cm$ to the left of centered. This way, the arm can begin a new attempt at inspection

from a location that is less likely to result in a failure.

### 6.8 Final Results

The ultimate goal of this work is to perform PD sensor inspections autonomously on multiple panels labeled with barcode stickers. An experimental setup was created in the ABS Engineering Center at Stevens Institute of Technology with two barcodes placed at the height used within substations. To successfully perform a complete inspection, each method described in this section must work together, resulting in Fig. 6.19 which shows ordered tasks for each of the four basic functions necessary for autonomous inspections.

Testing the complete automation in the real world proved difficult to quantify with many steps. Therefore, the robots view of each step is shown in Fig. 6.20 for two barcode stickers. Each process boots up at the starting location, and after the command is sent to automate, the robots handle the rest. The first waypoint was chosen to be close to the starting location for rapid testing of the alignment functionality and the starting location is out of the way within the lab space. Panel alignment occurs, reducing the distance and angle between the wrist camera and the panel, through motion of the UGV. Upon accptable alignment, the Kinova Gen3 arm begins to search for the barcode sticker. A pre-programmed position is used to begin the search, however small yaw adjustments are made and a final distance check for $40cm$ is done to ensure the camera has optimal viewing conditions. After noticing the barcode sticker was too centered, the arm returned to the travel configuration before moving the UGV forward slightly and then returning to the search position. This motion is captured by the cyan trail between Fig. 6.20d and Fig. 6.20e. After the final alignment, the arm believes it can successfully perform the complete inspection

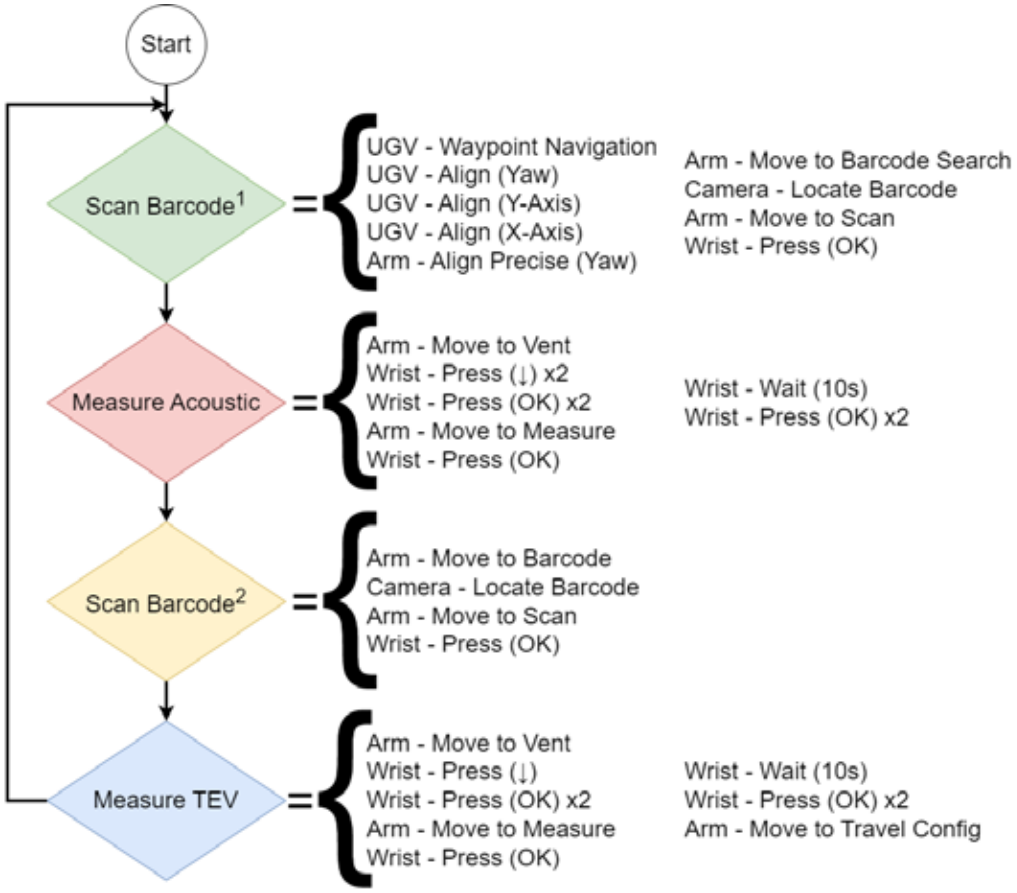
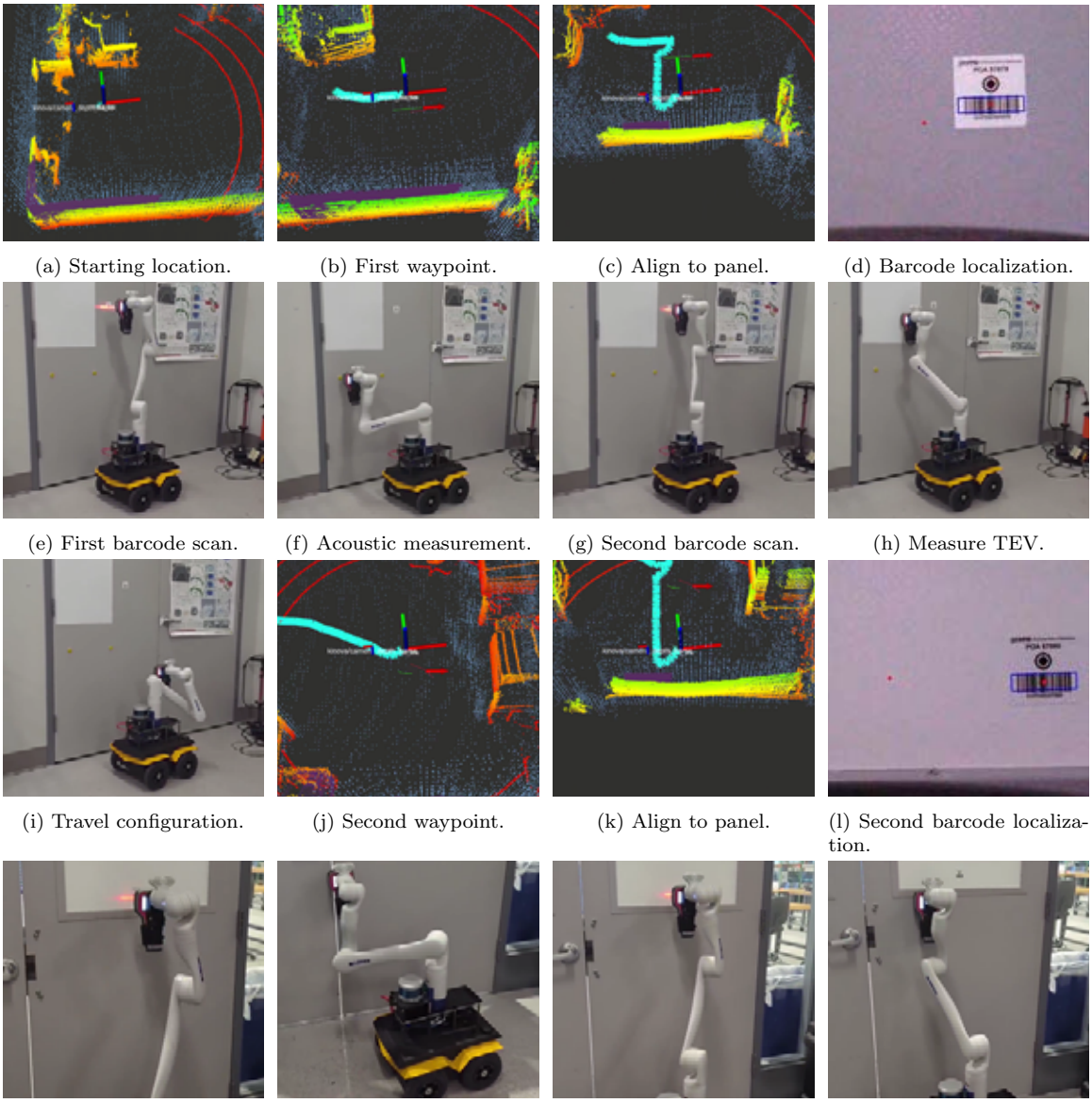



Figure 6.19: Simplified complete autonomous inspection tasks for each function based on each piece of hardware. Each task is ordered from left to right, top to bottom to ensure successful inspection.

task and begins scanning the barcode. Button pressing is unable to be sensed by the robots, and therefore does not have a representative figure.

Once the barcode has been scanned, the arm moves to the vent, located $41cm$ down from the sticker. To manage such large motions, the end-effector also moves $38cm$ to the left, ensuring the arms stays within the available workspace. The vent in the substation is approximately $80cm$ wide, with the barcode sticker $\approx 8cm$ right from center. Measuring acoustic data requires the PD sensor be close but not in contact with the vent, so the depth camera is used to place the sensor $1cm$ from the



(a) Starting location. (b) First waypoint. (c) Align to panel. (d) Barcode localization.

(e) First barcode scan. (f) Acoustic measurement. (g) Second barcode scan. (h) Measure TEV.

(i) Travel configuration. (j) Second waypoint. (k) Align to panel. (l) Second barcode localization.

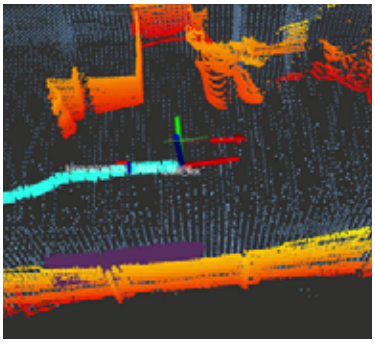
(m) Third barcode scan. (n) Acoustic measurement. (o) Fourth barcode scan. (p) Measure TEV.

Figure 6.20: Simplified Complete Automation Process tested in the lab space. Blue point cloud is the global reference map, multi-colored cloud is the Velodyne data used for localization, with color varying by height. A purple point cloud represents the depth data from the wrist camera. UGV and wrist axis are shown to represent the current location of each. A cyan path is included to show UGV navigation throughout the environment. Waypoints are indicated by the large red arrow. `https://robustfieldautonomylab.github.io/Pearson_ASME_2024.mp4`

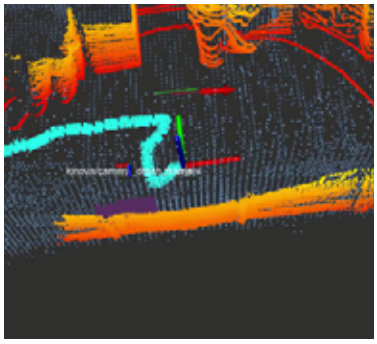
vent. Button pressing occurs to take the measurement and save the data. When the measurement is complete, the arm moves back to scan the barcode, having already identified its location relative to the UGV. Initial testing proved that searching for the barcode from that same position reduced the chance of failing to scan the barcode, therefore a new search begins for minor alignment. The simplified method to search for the barcode was used in these tests and performed as well as expected. With the arm aligned once again, the barcode is now scanned, using the depth camera to decide how close to move the end-effector to the panel. TEV measurement requires contact with the panel on any flat surface. A small downward offset from the barcode location was chosen to gather TEV data, and the depth camera once again decided how far the end-effector should move towards the panel. After more button pressing, the inpsection of the first panel is complete, and the arm returns to the travel configuration before moving to the second waypoint.

On the way over to the second waypoint, some obstacle avoidance occurred, resulting in the upward and downward motion seen in Fig. 6.20j. Some small maneuvers of the UGV occurred near the waypoint as it tried to meet the requirements necessary to declare itself arrived. Similar to the prior UGV and panel alignment, this one has a large distance to cover, showcasing the reliability of the alignment method by giving it tougher conditions than would be expected in the final product. The rest of the steps follow the same mechanics as the first barcode sticker inspection process, with the exception that the UGV no longer had to drive forward as barcode sticker was already sufficiently off center for complete inspection.

Another inspection was performed in a real substation, to showcase the complete expected behavior of our platform. This time, three panels were inspected consecutively. Each waypoint was given a large offset from the panels as seen in Fig. 6.21d and Fig. 6.21g even though it was possible to simply drive straight forward

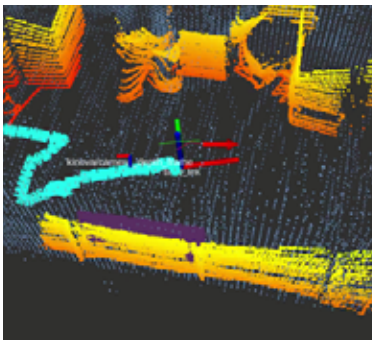


(a) UGV moves to the first waypoint.



(b) UGV aligns to panel.

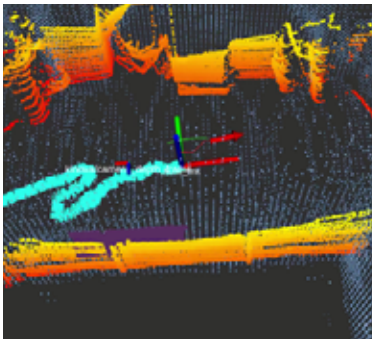

(c) Camera locates the barcode sticker.



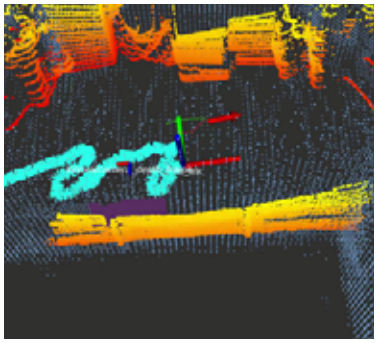
(d) UGV moves to the second waypoint.

(e) UGV aligns to the panel.

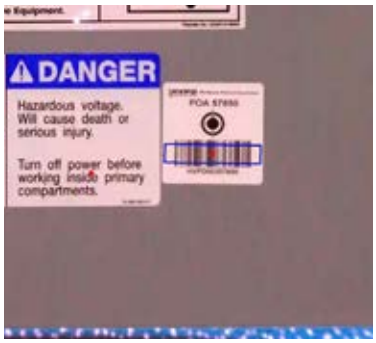

(f) Camera locates the second barcode sticker.



(g) UGV moves to the third waypoint.



(h) UGV aligns to the panel.




(i) Camera locates the third barcode sticker.

Figure 6.21: Simplified Complete Automation Process tested in a substation.

once the first alignment was made. This choice was made to showcase the reliability of the alignment methods used and ensure that obstacles would not pose an issue between inspections. Even with a drastic change in lighting and various other stickers on the panel, the barcode localization worked well with the "simplified" method in

the substation environment. Aside from booting up the programs and running the initial command, autonomous inspection was performed successfully.

## 6.9 Conclusion

Autonomous substation inspections will reduce failures and provide safer working environments for those delivering a crucial utility towards our way of life. While a simple handheld PD sensor might seem like an easy task to automate, many components were necessary to successfully perform autonomous inspections. Driving between inspection locations and avoiding obstacles, operating a robotic manipulator with precision and using both LiDAR and computer vision to localize within the surroundings were required for complete autonomy. A custom gripper with specialized button pressing electronics and hardware was created to interact with the PD sensor. While there were many methods to manage each of these tasks, our team chose specific robotic platforms ideal for the mobile manipulation in cluttered indoor environments. Each algorithm was tested for performance, and the optimal choice decided for the final version. A complete autonomous inspection was performed on hardware using the algorithms described in this dissertation.

While not explicitly mentioned in this dissertation, a few more features would be required for daily autonomous inspections. The PD sensor needs to be reset after gathering a dataset. This can be done by letting the sensor fall asleep, however a press of the $<$ key would be needed to go to the list of barcodes. Returning to a home waypoint for the Jackal to restart localization has been tested in the waypoint navigation work, however was not tested with the complete hardware setup. Tangentially, a charging dock would be required for the battery to be recharged between inspections, which would require a more accurate alignment method than

simple waypoint navigation. The current charging capabilities of the UGVs battery are insufficient for the power draw when everything is on, therefore a power saving mode would be necessary during the time when the robots are not in use. Due to some pitching of the wrist during arm motion, some barcode scans failed. One future solution would be to use a small vertical translation down across the barcode with the PD sensor actively trying to scan.

# Chapter 7
# Dissertation Conclusion

Prior works have focused on laying a strong foundation for autonomous inspection. By differentiating unknown and uncertain states in occupancy grid maps, implementing fast and reliable waypoint navigation on an unmanned ground vehicle and integrating a far reaching manipulator onto a compact UGV for precise inspection tasks. The culmination of these works resulted in an autonomous substation inspection routine that required many components working together. Successful autonomous inspections occurred both in the lab space and in one of the Con Edison indoor substations.

Milestone features were tested within the Robust Field Autonomy Lab space before scheduling a substation visit, such as the one seen in Fig. 7.1. Reliability varies among tested components of the inspection process. While barcode localization appear to be robust, slight variations occur in the $X$ axis when the opencv package identifies a barcode. With a tight tolerance, this can result in a "wiggle" as the arm tries to confirm the localization. One method that could resolve this issue would be to rely on a sweeping vertical motion for scanning the barcode. The implemented method attempts to scan the barcode during the approach from $40cm$ away, requiring highly accurate end-effector placement.

Precision of waypoint navigation has resulted in some failures to locate the barcode entirely. The $20cm$ wide view of the camera drastically limits the search range along the $Y$ axis. Considering that there is an implemented method to move the UGV slightly forward if the barcode is within the "dead zone", it seems practical to use the same system to move the UGV if the barcode is located too far to either side. However, that requires the camera to locate the barcode while out of the standard

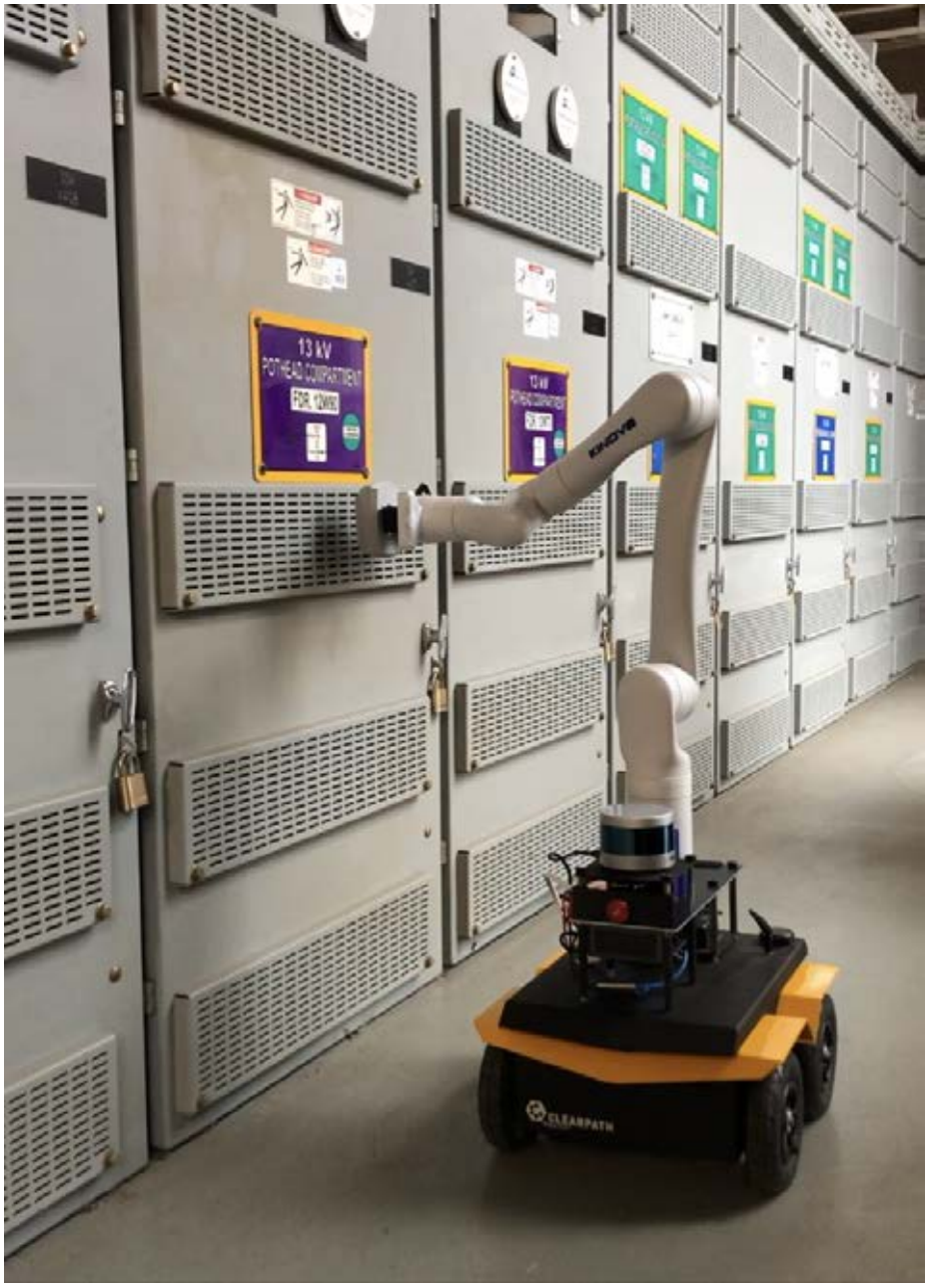



Figure 7.1: Mobile Manipulation in a Substation

viewpoint. One such method would be to move the camera $20cm$ to the left or right and attempt to locate the barcode from there. Given how high up the barcode is located, this may be impractical based on the total reach of the arm. However, there is definitely a method to increase the $Y$ axis range that is likely worth pursuing to

reduce the case of failures.

While only a few details could improve the robust behavior of the complete autonomous inspection program, there are some other difficulties that would arise from trying to make the entire system autonomous. Power is one of the main concerns. Pass through charging can result in irreparable damage to batteries, so the systems must be turned off when idle. Clearpath does offer a docking station, however our lab has yet to test the behavior associated with that hardware.

Sending a start command remotely would be sufficient for semi-autonomous substation inspections, however there is one major issue that our lab was unable to resolve. To choose the list of barcodes for the PD sensor to scan, a single "<" button press is required. This is only necessary to begin the inspection process, and there are no other times when that button needs to be pressed. Therefore, the gripper was designed to only fit two servos, for the "OK" and "∨" buttons. The PD sensor is also battery operated, and would need a method for charging as well.

## Bibliography

[1] E. Pearson, K. Doherty, and B. Englot, "Improving obstacle boundary representations in predictive occupancy mapping," *Robotics and Autonomous Systems*, p. 104077, 2022.

[2] E. Pearson and B. Englot, "A robust and rapidly deployable waypoint navigation architecture for long-duration operations in gps-denied environments," in *2023 20th International Conference on Ubiquitous Robots (UR)*, pp. 319–326, IEEE, 2023.

[3] E. Pearson, P. Szenher, C. Huang, and B. Englot, "Mobile manipulation platform for autonomous indoor inspection in low-clearance areas," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, ASME, 2023.

[4] E. Pearson, B. Mirisola, C. Murphy, C. Huang, C. O'Leary, F. Wong, J. Meyerson, L. Bonfim, M. Zecca, N. Spina, P. Szenher, S. Katari, S. Bhatt, T. Dohi, T. Colarusso, T. Gana, and B. Englot, "Robust autonomous mobile manipulation for substation inspection," in *Journal of Mechanisms and Robotics*, ASME, Under Review, 2024.

[5] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*, pp. 1–4, IEEE, 2011.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[7] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[8] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[9] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[10] D. Duberg and P. Jensfelt, "Ufomap: An efficient probabilistic 3d mapping framework that embraces the unknown," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6411–6418, 2020.

[11] N. Funk, J. Tarrio, S. Papatheodorou, M. Popović, P. F. Alcantarilla, and S. Leutenegger, "Multi-resolution 3d mapping with explicit free space representation for fast and accurate mobile robot motion planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3553–3560, 2021.

[12] K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 953–966, 2019.

[13] J. Wang and B. Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1003–1010, IEEE, 2016.

[14] K. Doherty, J. Wang, and B. Englot, "Bayesian generalized kernel inference for occupancy map prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3118–3124, IEEE, 2017.

[15] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[16] A. J. Weinstein and K. L. Moore, "Pose estimation of Ackerman steering vehicles for outdoors autonomous navigation," in *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 579–584, 2010.

[17] D. Wooden, M. Malchano, K. Blankespoor, A. Howardy, A. A. Rizzi, and M. Raibert, "Autonomous navigation for BigDog," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4736–4741, 2010.

[18] A. Chilian, H. Hirschmüller, and M. Görner, "Multisensor data fusion for robust pose estimation of a six-legged walking robot," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2497–2504, 2011.

[19] R. Wang, Y. Xu, M. A. Sotelo, Y. Ma, T. Sarkodie-Gyan, Z. Li, and W. Li, "A robust registration method for autonomous driving pose estimation in urban dynamic environment using lidar," *Electronics*, vol. 8, no. 1, p. 43, 2019.

[20] S. Ratz, M. Dymczyk, R. Siegwart, and R. Dubé, "Oneshot global localization: Instant lidar-visual pose estimation," in *2020 IEEE International conference on Robotics and Automation (ICRA)*, pp. 5415–5421, IEEE, 2020.

[21] M. Oelsch, M. Karimi, and E. Steinbach, "RO-LOAM: 3D reference object-based trajectory and map optimization in LiDAR odometry and mapping," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6806–6813, 2022.

[22] M. Labbe and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2661–2666, 2014.

[23] J. McDonald, M. Kaess, C. Cadena, J. Neira, and J. J. Leonard, "Real-time 6-dof multi-session visual slam over large-scale environments," *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1144–1158, 2013.

[24] M. Labbé and F. Michaud, "Multi-session visual slam for illumination-invariant re-localization in indoor environments," *Frontiers in Robotics and AI*, p. 115, 2022.

[25] P. Egger, P. V. Borges, G. Catt, A. Pfrunder, R. Siegwart, and R. Dubé, "Posemap: Lifelong, multi-environment 3d lidar localization," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3430–3437, IEEE, 2018.

[26] M. Zhao, X. Guo, L. Song, B. Qin, X. Shi, G. H. Lee, and G. Sun, "A general framework for lifelong localization and mapping in changing environment," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3305–3312, IEEE, 2021.

[27] H. Yin, Z. Lin, and J. K. Yeoh, "Semantic localization in BIM using a 3D LiDAR sensor," *arXiv preprint arXiv:2205.00816*, 2022.

[28] H. Blum, J. Stiefel, C. Cadena, R. Siegwart, and A. Gawel, "Precise robot localization in architectural 3d plans," *arXiv preprint arXiv:2006.05137*, 2020.

[29] B. Dzodzo, L. Han, X. Chen, H. Qian, and Y. Xu, "Realtime 2d code based localization for indoor robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 486–492, 2013.

[30] M. Mattamala, N. Chebrolu, and M. Fallon, "An efficient locally reactive controller for safe navigation in visual teach and repeat missions," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2353–2360, 2022.

[31] T. Krajník, F. Majer, L. Halodová, and T. Vintr, "Navigation without localisation: reliable teach and repeat based on the convergence theorem," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1657–1664, 2018.

[32] N. Piasco, D. Sidibé, C. Demonceaux, and V. Gouet-Brunet, "A survey on visual-based localization: On the benefit of heterogeneous data," *Pattern Recognition*, vol. 74, pp. 90–109, 2018.

[33] T. Krajník, P. Cristóforis, K. Kusumam, P. Neubert, and T. Duckett, "Image features for visual teach-and-repeat navigation in changing environments," *Robotics and Autonomous Systems*, vol. 88, pp. 127–141, 2017.

[34] T. B. . CIGRE, "Application of robotics in substations," *WG B3.47*, pp. 1–156, 2020.

[35] Y. Yamamoto and X. Yun, "Coordinating locomotion and manipulation of a mobile manipulator," in *[1992] Proceedings of the 31st IEEE Conference on Decision and Control*, pp. 2643–2648, IEEE, 1992.

[36] O. Khatib, "Mobile manipulation: The robotic assistant," *Robotics and Autonomous Systems*, vol. 26, no. 2-3, pp. 175–183, 1999.

[37] N. Kashiri, L. Baccelliere, L. Muratore, A. Laurenzi, Z. Ren, E. M. Hoffman, M. Kamedula, G. F. Rigano, J. Malzahn, S. Cordasco, *et al.*, "Centauro: A hybrid locomotion and high power resilient manipulation platform," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1595–1602, 2019.

[38] A. Jain and C. C. Kemp, "El-e: an assistive mobile manipulator that autonomously fetches objects from flat surfaces," *Autonomous Robots*, vol. 28, pp. 45–64, 2010.

[39] B. Graf, U. Reiser, M. Hägele, K. Mauz, and P. Klein, "Robotic home assistant care-o-bot® 3-product vision and innovation platform," in *2009 IEEE Workshop on Advanced Robotics and its Social Impacts*, pp. 139–144, IEEE, 2009.

[40] C. C. Kemp, A. Edsinger, H. M. Clever, and B. Matulevich, "The design of stretch: A compact, lightweight mobile manipulator for indoor human environments," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 3150–3157, IEEE, 2022.

[41] H. Engemann, S. Du, S. Kallweit, P. Cönen, and H. Dawar, "Omnivil—an autonomous mobile manipulator for flexible production," *Sensors*, vol. 20, no. 24, p. 7249, 2020.

[42] R. Bischoff, U. Huggenberger, and E. Prassler, "Kuka youbot-a mobile manipulator for research and education," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1–4, IEEE, 2011.

[43] J. Okamoto Jr, J. C. Adamowski, M. S. Tsuzuki, F. Buiochi, and C. S. Camerini, "Autonomous system for oil pipelines inspection," *Mechatronics*, vol. 9, no. 7, pp. 731–743, 1999.

[44] C. Balaguer, A. Giménez, J. M. Pastor, V. Padron, and M. Abderrahim, "A climbing autonomous robot for inspection applications in 3d complex environments," *Robotica*, vol. 18, no. 3, pp. 287–297, 2000.

[45] M. Jacobi, "Autonomous inspection of underwater structures," *Robotics and Autonomous Systems*, vol. 67, pp. 80–86, 2015.

[46] Y.-J. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Büyüköztürk, "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 731–747, 2018.

[47] S. Lu, Y. Zhang, and J. Su, "Mobile robot for power substation inspection: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 830–847, 2017.

[48] J. Huang, J. Wang, Y. Tan, D. Wu, and Y. Cao, "An automatic analog instrument reading system using computer vision and inspection robot," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 9, pp. 6322–6335, 2020.

[49] Y. Xiao, Y. Yan, Y. Yu, B. Wang, and Y. Liang, "Research on pose adaptive correction method of indoor rail mounted inspection robot in gis substation," *Energy Reports*, vol. 8, pp. 696–705, 2022.

[50] R. C. C. d. M. Santos, M. C. Silva, R. L. Santos, E. Klippel, and R. A. Oliveira, "A mobile robot based on edge ai," in *Anais do L Seminário Integrado de Software e Hardware*, pp. 191–202, SBC, 2023.

[51] K. Tan, W. Luo, and T. Li, "An intelligent and collaborative substation inspection system," in *2021 IEEE Sustainable Power and Energy Conference (iSPEC)*, pp. 4120–4124, IEEE, 2021.

[52] P. Dandurand, J. Beaudry, C. Hébert, P. Mongenot, J. Bourque, and S. Hovington, "All-weather autonomous inspection robot for electrical substations," in *2022 IEEE/SICE International Symposium on System Integration (SII)*, pp. 303–308, IEEE, 2022.

[53] R. Luo, M. Tang, and D. Liang, "On-site partial discharge test of medium voltage switchgear by time of arrival method," in *2014 17th International Conference on Electrical Machines and Systems (ICEMS)*, pp. 1023–1028, IEEE, 2014.

[54] C. Walton, S. Carter, M. Michel, and C. Eastham, "Avoidance of mv switchgear failure case studies of on-line condition monitoring," in *CIRED 2009-20th International Conference and Exhibition on Electricity Distribution-Part 1*, pp. 1–4, IET, 2009.

[55] P. Dehghanian, T. Popovic, and M. Kezunovic, "Circuit breaker operational health assessment via condition monitoring data," in *2014 North American Power Symposium (NAPS)*, pp. 1–6, IEEE, 2014.

[56] J. Köhler, A. Pagani, and D. Stricker, "Detection and identification techniques for markers used in computer vision," in *Visualization of Large and Unstruc-*

*tured Data Sets-Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[57] C. Creusot and A. Munawar, "Real-time barcode detection in the wild," in *2015 IEEE winter conference on applications of computer vision*, pp. 239–245, IEEE, 2015.

[58] V. Patidar and R. Tiwari, "Survey of robotic arm and parameters," in *2016 International conference on computer communication and informatics (ICCCI)*, pp. 1–6, IEEE, 2016.

[59] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[60] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2012.

[61] I. Ko, B. Kim, and F. C. Park, "Randomized path planning on vector fields," *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1664–1682, 2014.

[62] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! task constructor for task-level motion planning," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 190–196, IEEE, 2019.

[63] A. Abe, "Trajectory planning for flexible cartesian robot manipulator by using artificial neural network: numerical simulation and experimental verification," *Robotica*, vol. 29, no. 5, pp. 797–804, 2011.

[64] E. Brzozowska, O. Lima, and R. Ventura, "A generic optimization based cartesian controller for robotic mobile manipulation," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2054–2060, IEEE, 2019.

[65] Y. Chen, X. Luo, B. Han, Q. Luo, and L. Qiao, "Model predictive control with integral compensation for motion control of robot manipulator in joint and task spaces," *IEEE Access*, vol. 8, pp. 107063–107075, 2020.

[66] A. H. Khan, S. Li, and X. Luo, "Obstacle avoidance and tracking control of redundant robotic manipulator: An rnn-based metaheuristic approach," *IEEE transactions on industrial informatics*, vol. 16, no. 7, pp. 4670–4680, 2019.

[67] K. Merckaert, B. Convens, C.-j. Wu, A. Roncone, M. M. Nicotra, and B. Vanderborght, "Real-time motion control of robotic manipulators for safe human–robot coexistence," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102223, 2022.

[68] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*, pp. 524–530, IEEE, 2012.

[69] A. Melkumyan and F. T. Ramos, "A sparse covariance function for exact gaussian process inference in large datasets," in *Twenty-first international joint conference on artificial intelligence*, 2009.

[70] M. G. Jadidi, J. V. Miro, and G. Dissanayake, "Warped gaussian processes occupancy mapping with uncertain inputs," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 680–687, 2017.

[71] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[72] P. Sodhi, B.-J. Ho, and M. Kaess, "Online and consistent occupancy grid mapping for planning in unknown environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7879–7886, IEEE, 2019.

[73] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012.

[74] S. Kim and J. Kim, "Gpmap: A unified framework for robotic mapping based on sparse gaussian processes," in *Field and service robotics*, pp. 319–332, Springer, 2015.

[75] S. Kim, J. Kim, *et al.*, "Recursive bayesian updates for occupancy mapping and surface reconstruction," *Australasian Conference on Robotics and Automation*, 2014.

[76] L. Gan, R. Zhang, J. W. Grizzle, R. M. Eustice, and M. Ghaffari, "Bayesian spatial kernel smoothing for scalable dense semantic mapping," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 790–797, 2020.

[77] W. R. Vega-Brown, M. Doniec, and N. G. Roy, "Nonparametric bayesian inference on multivariate exponential families," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[78] S. T. O'Callaghan and F. T. Ramos, "Continuous occupancy mapping with integral kernels," in *Twenty-fifth aaai conference on artificial intelligence*, 2011.

[79] K. S. Shankar and N. Michael, "Mrfmap: Online probabilistic 3d mapping using forward ray sensor models," *arXiv preprint arXiv:2006.03512*, 2020.

[80] F. Ramos and L. Ott, "Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[81] V. Guizilini and F. Ramos, "Large-scale 3d scene reconstruction with hilbert maps," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3247–3254, IEEE, 2016.

[82] V. Guizilini and F. Ramos, "Learning to reconstruct 3d structures for occupancy mapping from depth and color information," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1595–1609, 2018.

[83] S. Guo and N. A. Atanasov, "Information filter occupancy mapping using decomposable radial kernels," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7887–7894, IEEE, 2019.

[84] A.-A. Agha-Mohammadi, E. Heiden, K. Hausman, and G. Sukhatme, "Confidence-rich grid mapping," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1352–1374, 2019.

[85] D. Fridovich-Keil, E. Nelson, and A. Zakhor, "Atommap: A probabilistic amorphous 3d map representation for robotics and surface reconstruction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3110–3117, IEEE, 2017.

[86] Y. Kwon, B. Moon, and S.-E. Yoon, "Adaptive kernel inference for dense and sharp occupancy grids," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4712–4719, IEEE, 2020.

[87] M. G. Jadidi, J. V. Miró, R. Valencia, and J. Andrade-Cetto, "Exploration on continuous gaussian process frontier maps," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6077–6082, IEEE, 2014.

[88] M. Ghaffari Jadidi, J. Valls Miro, and G. Dissanayake, "Gaussian processes autonomous mapping and exploration for range-sensing mobile robots," *Autonomous Robots*, vol. 42, no. 2, pp. 273–290, 2018.

[89] T. Teixeira, F. Mutz, K. S. Komati, L. Veronese, V. B. Cardoso, C. Badue, T. Oliveira-Santos, and A. F. De Souza, "Memory-like map decay for autonomous vehicles based on grid maps," *arXiv preprint arXiv:1810.02355*, 2018.

[90] K. Koide, J. Miura, and E. Menegatti, "A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419841532, 2019.

[91] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[92] E. W. Dijkstra, "A note on two problems in connexion with graphs:(numerische mathematik, 1 (1959), p 269-271)," 1959.

[93] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[94] L. Kneip, F. Tâche, G. Caprari, and R. Siegwart, "Characterization of the compact hokuyo urg-04lx 2d laser range scanner," in *2009 IEEE International Conference on Robotics and Automation*, pp. 1447–1454, IEEE, 2009.

[95] J. Lambert, A. Carballo, A. M. Cano, P. Narksri, D. Wong, E. Takeuchi, and K. Takeda, "Performance analysis of 10 models of 3d lidars for automated driving," *IEEE Access*, vol. 8, pp. 131699–131722, 2020.

[96] J. Laconte, S.-P. Deschênes, M. Labussière, and F. Pomerleau, "Lidar measurement bias estimation via return waveform modelling in a context of 3d mapping," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8100–8106, IEEE, 2019.

[97] R. Angelari, "A deterministic and random error model for a multibeam hydrographic sonar system," in *OCEANS'78*, pp. 48–53, IEEE, 1978.

[98] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun, "Lidarsim: Realistic lidar simulation by leveraging the real world," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11167–11176, 2020.

[99] J. P. Espineira, J. Robinson, J. Groenewald, P. H. Chan, and V. Donzella, "Realistic lidar with noise model for real-time testing of automated vehicles in a virtual environment," *IEEE Sensors Journal*, vol. 21, no. 8, pp. 9919–9926, 2021.

[100] Y. Zhong and H. Peng, "Real-time semantic 3d dense occupancy mapping with efficient free space representations," *arXiv preprint arXiv:2107.02981*, 2021.

[101] S. Macenski, D. Tsai, and M. Feinberg, "Spatio-temporal voxel layer: A view on robot perception for the dynamic world," *International Journal of Advanced Robotic Systems*, vol. 17, no. 2, p. 1729881420910530, 2020.

## Vita

**Erik Pearson**

### Education

PH.D. in Mechanical Engineering — August 2016 - January 2024
Stevens Institute of Technology, Hoboken, NJ

M.E. in Mechanical Engineering — August 2016 - January 2024
Stevens Institute of Technology, Hoboken, NJ

B.S. in Mechanical Engineering — August 2011 - June 2015
Massachusetts Institute of Technology, Cambridge, MA

B.S. in Mathematics — August 2011 - June 2015
Massachusetts Institute of Technology, Cambridge, MA

### Work Experience

New Valence Robotics, Cambridge, MA
Product Engineer — June 2015 - August 2016

### Publications

**Erik Pearson**, Kevin Doherty and Brendan Englot, " Improving Obstacle Boundary Representations in Predictive Occupancy Mapping," *Robotics and Autonomous Systems (RAS),* vol. 153, Article 104077, July 2022.

**Erik Pearson** and Brendan Englot, "Simplified Multi-Session Robot Navigation using Waypoints," *Ubiquitous Robots (UR),* pp. 319-326, 2023.

**Erik Pearson**, Paul Szenher, Christine Huang and Brendan Englot, "Mobile Manipulation Platform for Autonomous Indoor Inspection in Low-Clearance Areas," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE),* 2023.

**Erik Pearson**, Benjamin Mirisola, Cameron Murphy, Christine Huang, Connor O'Leary, Franklin Wong, Julia Meyerson, Luisa Bonfim, Matthew Zecca, Noah Spina, Paul Szenher, Shalemuraju Katari, Shiv Bhatt, Takumasa Dohi, Thomas Colarusso, Thomas Gana, and Brendan Englot, "Robust Autonomous Mobile Manipulation for Substation Inspection," *Under Review - Journal of Mechanisms and Robotics,* 2024.

# Appendix A
# Further Studies of Occupancy Grid Mapping

## A.1 Background - Sensor Modeling

To create the most effective inference based mapping algorithm, we must first understand the limitations of the incoming data, which depends on the sensor used. Occupancy mapping typically uses range data sourced from sonar, radar, lidar or stereo camera sensors. Each source has different capabilities and limitations that work best in unique environments. For example, turbid underwater conditions severely limits radar, lidar and stereo camera sensors whereas sonars are able to collect data as intended.

Inference based occupancy grid mapping results will vary depending on the sensor used. However, the inferences and predictions were created to deal with both sparsity and noise. Therefore, to create a better inference model we will first look at the sources of noise from each sensor and their affects on the data.

One common source of noise is surface material. Extensive testing shows that reflectivity affects the residual error of range data for LiDAR sensors [94, 95]. This is one source of noise in the $\Delta r$ direction.

However, not all sources of noise cause errors in the $\Delta r$ direction. For instance, incidence angle of sensed surface causes data deflections in the $\Delta\theta$ direction [96]. This sensor bias was tested thoroughly on multiple LiDAR sensors which had varying degrees of error that scaled with both range and incidence angle.

LiDAR sensors are also affected by internal heating. A study was performed where the environment was kept constant while the sensor continuously gathered data

producing an error in the $\Delta r$ direction [94].

Weather conditions can affect range data too. Precipitation and opaqueness will affect LiDAR and stereo camera sensors while underwater conditions can vastly change the speed of sound and produce excessive pulse stretching [97]. These conditions are common enough that there are multiple active attempts to recreate such weather conditions in simulation [98, 99].

While there are many sensor options available for gathering range data, they all appear to produce errors in the $\Delta r,\ \Delta\theta$ & $\Delta\phi$ directions. Each sensor has its own unique error magnitude, however they all appear to scale with depth [95].

Assuming that errors are centered around the truth, we can define sensor noise as a set of Gaussian functions in each direction $\mathcal{G}_i(0, \sigma_i)\ \forall\ i \in \{r,\ \theta,\ \phi\}$. While each $\sigma_i$ is likely to have a different value, there is a common trait. The error increases proportionally with range. Some sensors, such as the Velodyne Puck, have a small increase in error caused by range while other sensors have more significant errors.

## A.2 Common Modifications to Training Data

### A.2.1 Modifying Occupied Space Training Data

Training data used to define Occupied space often comes in the form of a Point Cloud where each point represents a "hit" on a object around the sensor. For most simple sensor setups, the data is gathered from a single source which indicates a uniform angular distribution. While the distributions are generally dense enough to mimic continuous representation, that only holds for data gathered close to the sensor. The chord length between two neighboring beams from the sensor increases based on their distance from the sensor. The resulting data distribution from this physical phenomena has dense data near the sensor with increasingly sparse data the further

away it occurs.

Highly dense data is unnecessary for occupancy prediction and will actually be detrimental for computational complexity. Therefore, down-sampling is commonly used the reduce regions of dense data while maintaining regions of sparse data without generalization. Nearly all inference based occupancy grid mapping algorithms employ this method. Another common method that modifies training data is setting a distance limit from the sensor, where data beyond that range is excluded from the training data set. By setting such a limit, an algorithm further reduces the data to train on, while also specifically targeting data that is less accurate.

### A.2.2 Modifying Free Space Training Data

There are two common choices for the training data used to estimate Free Space, either a distribution of points between the sensor and each "hit", or the ray itself. However, most algorithms that focus on Free Space representation also "clean" the data before predicting with it. One common method is to reduce the training data by an offset from the "hit" that defines Occupied Space. While this separates Free from Occupied in the radial direction, the orthogonal directions are ignored. This naive method only creates an accurate representation of the data if the ray from sensor origin to "hit" is perpendicular to the surface found via the "hit". As the angle between the surface and the ray changes closer to parallel, the naive method breaks down. Rays that are nearly parallel to a surface are called glancing rays. Glancing rays present an issue with current inference based mapping methods as the naive method is unable to handle the boundary between occupied and free in the non-radial directions. Two methods have been found to reduce the error caused by this issue.

The bilinearly weighted ray method proposed in [100] manages to handle the

majority of glancing ray cases with minimal additional computational complexity. However, there are scenarios where glancing rays will still affect the inference result. The other method proposed in [1] chose to simply reduce rays that were considered glancing. Reducing the length of a glancing ray ensured that there was no overestimation from glancing rays, but also reduced some true estimation. Additional computation complexity was required to reduce the glancing ray by checking for nearby data along the length of the ray.

While setting a maximum range for occupied data is beneficial for reducing less accurate data, a choice must be made whether to use that data to build free space training data. Some inference based algorithms consider data beyond the max range to have never existed, while others perform free space analysis on it before excluding everything beyond the max range. The latter method ensures that free space will be predicted accurately up to the boundary of max range, instead of leaving a gap in the data as the prior method would.

## A.3 Proposed Changes

### A.3.1 Modifying the Kernel

Based on our understanding of the noise produced by sensors and with inspiration from AKIMap [86], a new inference based mapping algorithm was devised. A variable kernel length based on distance from the sensor resulted in the Linear Depth Kernel OctoMap (LDKMap). Adjusting the kernel length alone would produce excessive estimation, however we found a unique solution. The sparse kernel equation (3.11) can be integrated to find the total area under the curve seen in 3.2 by using the limits of $[-l, l]$ for the distance $d$ as seen in equation (A.1).

$$\int_{-l}^{l} \sigma \left[ \frac{2 + \cos(2\pi \frac{d}{l})}{3} (1 - \frac{d}{l}) + \frac{1}{2\pi} \sin(2\pi \frac{d}{l}) \right] dd = \frac{4\sigma l}{3} = \mathcal{K}_A \tag{A.1}$$

The area under the curve is directly proportional to the kernel length $l$ and the kernel strength $\sigma$. Therefore, to maintain fair inference while adjusting the kernel length, LDKMap uses a constant kernel area $\mathcal{K}_A$, making kernel strength inversely proportional to kernel length. As the region of influence increases farther from the sensor, the strength of our certainty about that data decreases.

To implement this feature into the code, the variables used for $l$ and $\sigma$ were replaced with *kern_area* and *len_scale*. The length scale factor defined the rate of linear increase of kernel length relative to the distance a node is from the sensor while the kernel area was treated as a constant used to calculate $\sigma$ during each prediction as seen in equation (A.2)

$$\sigma_i = \frac{3\mathcal{K}_A}{4l_i} \tag{A.2}$$

#### A.3.1.1 Pruning and Splicing

After establishing a large range of resolutions for Occupancy Grid Mapping in prior work, a new goal was the include significantly smaller resolutions without increasing computational complexity. Multiple resolutions have been effective at reducing the number of nodes to iterate over when using the map for inspections. The most common method to enable multiple resolutions is pruning, where eight map nodes of the same occupancy type are pruned into a single node with double the length, width and height. However, pruning alone has limitations, such as only increasing node size and requiring all occupancy calculations to occur at the smallest resolution.

Two steps were considered simultaneously to meet the new criteria. The first

**Algorithm A.1:** Linear Depth Kernel (LDK) Inference

**Data:** Training data: $\mathcal{X}, \mathcal{Y}$; Query point: $x_*$

**Function** LDK($\mathcal{X}, \mathcal{Y}$, $x_*$)**:**

- **for** *each* $(x_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ **do**
  - **if** $y_i = 0$ **then**
    - Compute $\hat{x_i} \leftarrow x_{free}$ from Eq. (3.9)
  - **else**
    - $\hat{x_i} \leftarrow x_i$
  - $k_i \leftarrow k(x_*, \hat{x_i})$ Sparse kernel, Eq. (3.11) (A.2)
  - $\alpha_* \leftarrow \tau\alpha_* + k_i y_i$
  - $\beta_* \leftarrow \tau\beta_* + k_i(1 - y_i)$
- **return** $\alpha_*, \beta_*$

requirement was an inverse function of prune, which we called "splice". Given that pruning occurs when occupancy states are the same, splice was designed to dissect a node into 8 sub-nodes based on conflicting occupancy data. The uncertain state can be used to easily ascertain conflicting data and assign a threshold. With splice implemented, the nominal resolution was set to be a middle resolution, with the ability to prune to larger resolutions and splice to smaller ones.

### A.3.2 Time Varying Data

Dynamic obstacles often occur in real world data gathering. However, most mapping algorithms take snapshots and treat that data as absolute. An alternative approach decays older data so that newer data is seen as more current [89, 101]. This feature was also implemented into LDKMap to enable rapid changes between occupied and free space with a slightly modified version of the BGK algorithm seen in algorithm A.1 with $\tau$ representing the decay factor producing an exponential decay each time new data occurs.

### A.4 Difficulties with Implementation

As expected when considering depth variant kernel length, far more calculations were required to process a single point cloud. While the varying kernel length may produce slightly more accurate results, the drastic decrease in frequency inhibited the potential for more populated and accurate occupancy grid maps. This was another incentive to build out the splicing system with the hopes of recouping some time back from the excess computations.

While the concept of splicing seems to be a clear opposite to pruning, there are some unique limitations. The two most difficult concepts to address were the requirements for splicing and how to handle the data already condensed into the current node. While the uncertain state does instantly qualify a node for splicing, that state alone may be a subset of the nodes which should be spliced. For example, an unknown node with occupancy $p = 0.6$ would represent a mid-sized node with some likelihood of being occupied but there is not enough data to decide that yet. However, if the node is spliced, the subset of the node where the occupied data occurs, may be sufficient to deem a node as fully occupied while the rest of the space is still considered unknown. Therefore, should the partially unknown node be spliced? Based on the lower level data, the answer is yes. However, if all the unknown nodes are spliced, then there's no point in starting at the middle level. Multi-resolution kernel inference has yet to be studied extensively, as all prior work used the same size nodes for occupancy predicts before pruning based on similarities. A minor but consequential issue is how to deal with kernel length and strength, as well as the $\gamma$ threshold for unknown. Larger nodes can be treated as a composite of lower nodes, resulting in a significantly larger unknown threshold. Using that assumption, the kernel length should also represent a larger region, but the kernel strength should

not be lowered for such a large region, therefore we have to use a multiple of the original kernel area $\mathcal{K}_A$ as well. This results in varying estimations as they are taken at different resolutions throughout the occupancy grid map.

The other major difficultly is how to treat occupancy data when splicing. Using the kernel inference, training data is compressed down into either free or occupied weights for each node. When pruning, the occupancy data can be added together, to retain all unique information inside the node while producing something of similar likelihood. However, splicing needs to find a way to separate out data which has already been condensed. During implementation, we attempted to check for splicing before updating the kernel within in each node. After the inference was calculated, we would check for the uncertain state using the newly acquired data with the prior node data. If the result was an uncertain state, splicing would occur where the prior node data was split evenly in 8 ways before disseminating those values down to each node below. Then, each new node was predicted with the current training data. The process would continue until no cells were defined as uncertain or the smallest resolution had been reached. As mentioned earlier, when predicting with multiple resolutions, thresholds would vary that cause unsatisfactory results. For example, a large enough node would have a large unknown threshold that the uncertain state would be hidden from view. Alternatively, the uncertain threshold, which represents a direct ratio, might too big to function correctly with the larger data within a mid sized node, which would result in either occupied or free designation even though there is conflicting data.

Adding the decaying feature to the kernel was simple and required minimal calculations, but did not change the functionality of the original mapping algorithm sufficiently to describe in a full paper.