Multistage Localization for High Precision Mobile Manipulation Tasks

Christopher James Mobley

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Mechanical Engineering

Tomonari Furukawa
Brian Lattimer
Kevin Kochersberger

Blacksburg, Virginia

Keywords: Autonomous Navigation, SLAM, Visual Servoing, Mobile Manipulation, Computer Vision, State Machine

Multistage Localization for High Precision Mobile Manipulation Tasks

Christopher James Mobley

(ABSTRACT)

This paper will show the general framework necessary in order to solve two main problems preventing the development and widespread adoption of automation and robotics in construction (ARC). These problems include the fact that typical construction sites tend to be unstructured and are continuously evolving versus the highly controlled environments found in manufacturing. Also, the relationship between the part and manipulator has been reversed, causing increased complexity not seen in manufacturing environments where the part arrives at a fixed manipulator. The techniques that will be presented allow systems to create a 2-D map of their environment, localize themselves and complete the task(s) assigned. After localizing an augmented reality (AR) tag at the work site, the system is able to use a priori knowledge to localize points of interest (POIs) and complete several different types of operations achieving an accuracy of approximately $\pm$ 2 mm based on a multifaceted computer vision approach with only a USB webcam.

Acknowledgments

Insert Acknowledgement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Unlike the substantial benefits seen in the manufacturing industry through automation and robotics, automation and robotics in construction (ARC) has lagged far behind in adoption [**Balaguer**]. Consequently, when compared with other industries, construction has seen a significant decrease in productivity, as well as an increase in workplace injuries/ fatalities over the last several decades [**Rojas2003**]. While several technical complexities inherent in construction have hindered the development and adoption of field construction robots [4], through the capitalization of advances made by other industries, ARC can quickly close this gap. Thereby allowing dangerous and or mundane repetitive tasks to be accomplished autonomously. Thus, causing an increase in productivity and a decrease in workplace injuries/fatalities [1]. However, ARC faces two unique challenges when compared to other industries. Unlike manufacturing environments, which are tightly controlled, typical construction sites tend to lack structure and are continuously evolving. In addition, the reversal in relationship between the part and manipulator has dramatically increased the complexity of the problem to be solved [**Feng2015**]. Instead of the part appearing at a fixed manipulator, the manipulator must now move to and localize itself with respect to the part. The remainder of this paper is structured as follows: In Section 2, the author's technical approach is outlined, with particular focus on problem two, and experimental results are shown. Conclusions are then drawn and future work discussed in Section 3.

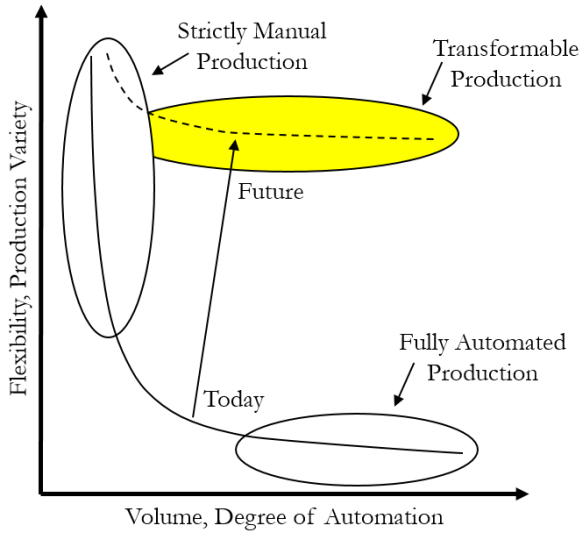


Figure 1.1: Vision for future transformable production systems.

1.2 Objectives

Insert Text Regarding the Objectives of This Work.

1.3 Summary of Original Contributions

Insert Text Regarding the Originals Contributions Presented in this Work.

1.4 Outline

Insert Text Outlining the Upcoming Chapters.

Chapter 2

Literature Review

2.1 Localization and Mapping For Autonomous Mobile Manipulators in Manufacturing and Construction

Insert Text Summarizing Current SLAM and Localization Techniques Used.

2.2 Task Association and A Priori Knowledge for Mobile Manipulators

Insert Text Summarizing Current Methods Used to Associate Mobile Manipulator to a Specific Task and How Prior Knowledge is Conveyed About the Task to be Performed.

2.3 Feature Localization Techniques for Mobile Manipulators

Insert Text Summarizing Current Methods Used to Localize a Feature and Have a Mobile Manipulator Perform a Set Operation.

2.4 Summary

Insert Text Summarizing This Chapter and Transiting to the Next.

Chapter 3

Fundamentals of Autonomous Robotics

The purpose of this chapter is to briefly explain all concepts needed to understand the work presented in the chapters thereafter. The following subsection will explain the basic concepts of the Robot Operating System (ROS), Simultaneous Localization and Mapping (SLAM), Localization and Path Planning, as well as Manipulation. In addition, Task Execution using State Machines and the specific Computer Vision Algorithms implemented will be expounded upon.

3.1 ROS Concepts

The Robot Operating System (ROS) [1] is an open-source set of software libraries and tools that aim to simplify the task of creating robotics applications that can be used across a wide variety of platforms. ROS was originally developed in 2007 by Willow Garage as an extension of switchyard, a collection of robotics software developed by Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot (STAIR) and Personal Robotics (PR) projects. The first distribution of ROS, Box Turtle, was released in 2010. ROS currently has had ten major releases. The most current being Kinetic Kame Turtle, which was released on 23MAY16. In addition ROS boasts tens of thousands of users around the world ranging from hobbyists and researcher to the commercial and industrial industries, as well as hundreds of packages which provide everything from hardware drivers to algorithms for autonomous navigations and manipulation [2]. ROS's large support base act as a force multiplier allows individuals, Labs, or company to concentrate on one particular aspect while capitalizing on work that has already been done.

3.1.1 ROS Communication

ROS uses a name server, called the ROS Master, to maintain a list of nodes and available topics. Nodes communicate with the master server using the XML-RPC protocol. While peer-to-peer communications generally use TCP/IP sockets through the TCPROS protocol. Figure 3.1 shows a simple diagram of two ROS nodes communicating with messages and service topics. In addition to the concepts of messages and services, ROS also employs actions. Actions are similar to service calls, but are designed for long duration tasks that are capable of providing feedback. These communication interfaces provide ROS a great deal of flexibility for robotic applications [3], [4].

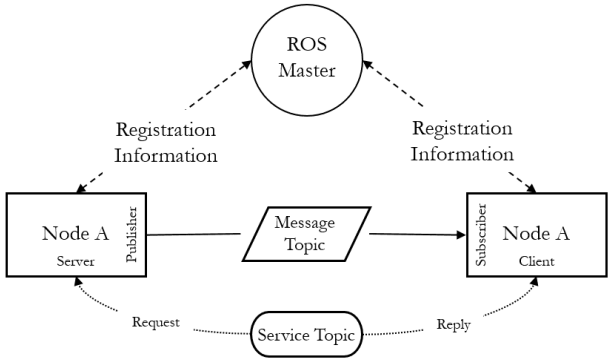


Figure 3.1: Simple ROS Node Flowchart. Adapted from [4].

3.1.2 Rigid Body Transformations

Figure 3.2 shows a purple dot, which represent a point in space. The dot's coordinates in frames a and b are different. A rigid body transform, which can be performed using Equation 3.1, can be used to convert one set of coordinates to another coordinate frame.

$$x_a = T_b^a x_b \tag{3.1}$$

where T_b^a is equal to Equation 3.2

$$\begin{bmatrix} R_b^a & t_b^a \\ 0^T & 1 \end{bmatrix} \tag{3.2}$$

where R_b^a is the rotation matrix, which performs the rotation part of moving frame b into alignment with frame a and t_b^a is the translation matrix, which performs the translation part of moving frame b origin to frame a [5].

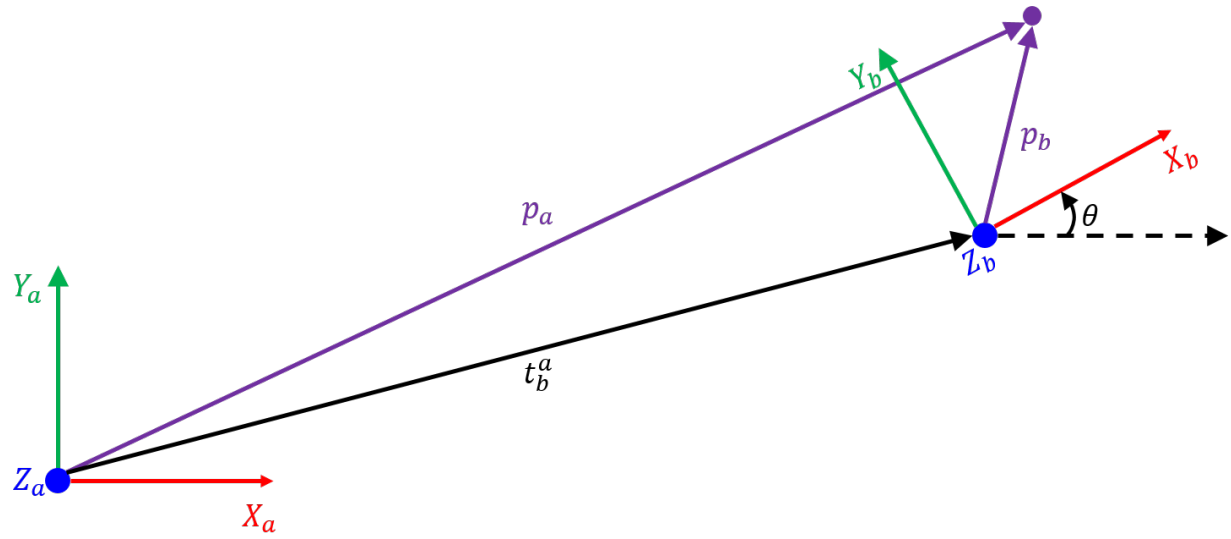


Figure 3.2: Conversion from Coordinate Frame A to B.

A robotic system, such as Clearpath Robotics' Husky and Fetch Robotics' Fetch shows in Figure 3.3, typical has many three dimensional coordinate frames that change over time as the robot performs different functions. ROS's TF [6] and TF2 [7] package keep track of coordinate frames and allow for data to be easily convert between these coordinate frames using rigid body transforms. These coordinate frames, as well as robot's links/joints they correspond to, and the rigid transformations between them are set up in the Robot's Unified Robot Description Format (URDF) File [8].

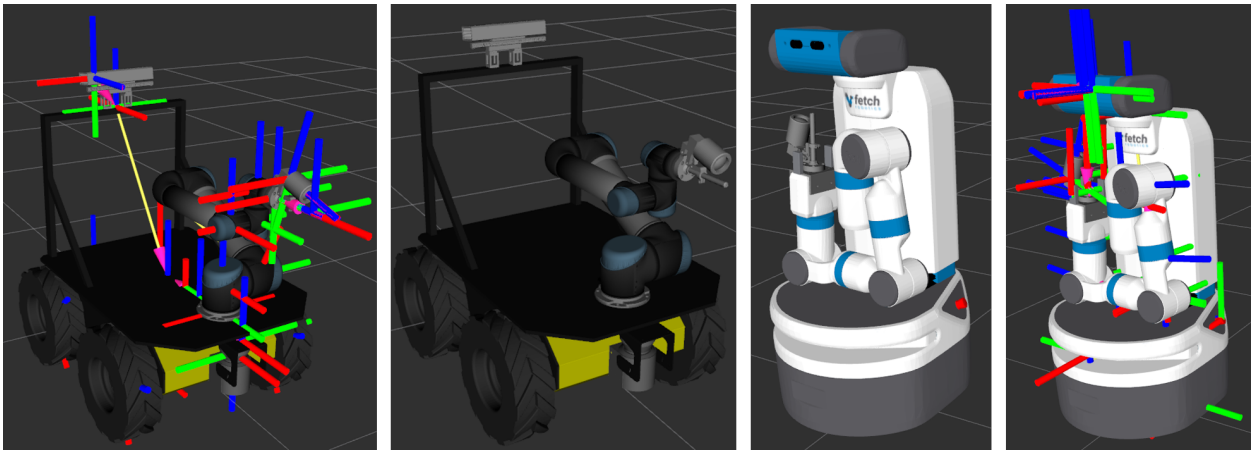


Figure 3.3: Robot Model with TFs.

3.2 Simultaneous Localization and Mapping Concepts

In order for a mobile robot to operate and perform intricate tasks within a complex, GPS-denied environments, such as that of manufacturing facilities, without modifying said environment, the robot must be able to create an accurate map of its environment while simultaneously localizing itself within this map using only on-board sensors. Simultaneous localization and mapping (SLAM) is the problem of building and/or updating a map of an unknown environment while simultaneously localizing the robot within this map [9]. SLAM was pioneered in the early 1990s by Hugh F. Durrant-Whyte and John J. Leonard [10], who based their work on research done by Smith and P. Cheeseman in the mid to late–1980s [11], [12].

Several techniques exist to solve the SLAM problem. Most of these techniques can be categorized into two main paradigms: filtering and optimization-based smoothing [13], [14]. Filtering techniques model the SLAM problem as an incremental state estimation, where the state of the system is composed of the robot's current pose and the map. These estimates are refined at each step by incorporating current sensor measurements. Due to their incremental nature, these SLAM techniques are typically referred to as on-line SLAM approaches. Popular filtering SLAM techniques include the extended Kalman filter, particle filter, and information filters. Filtering SLAM techniques have been used widely in past years due to their ability to model different sources of noise and their effect on sensor measurements. However, in recent years optimization-based smoothing techniques have proven to be more efficient, scalable, and robust than that of filtering techniques [14]. Unlike filtering techniques, optimization-based smoothing techniques estimate the robot's entire trajectory and the map. Due to the fact that the final map is based off the robot's entire trajectory and world features instead of the most recent pose and map, these techniques are known as full SLAM approaches. These SLAM techniques incorporate a graph-based structure, where graph nodes represent the robot's pose and world features, while edges represent a spatial constraint relation between two robot poses given by sensor measurements [15]. The graph is optimized using error minimization techniques, such as least-squares, in order to refine the robot's trajectory and map.

In addition to a variety of techniques which can be used to solve the SLAM problem, a wide range of sensors can also be used. Typically sensors include that of a LIDARs, stereo cameras, monocular camera, and RGB-D sensors.

Figure 3.4 shows the generic framework to solving the SLAM problem. In the front-end, raw sensor inputs are processed in order to extract features and perform scan matching; so that, necessary parameters and/or constraints, as well as systems state can be estimated. The systems states and necessary parameters and/or constraints are sent to the back-end of the SLAM algorithm where the systems state is refined and returned based on the parameters and/or constraints.

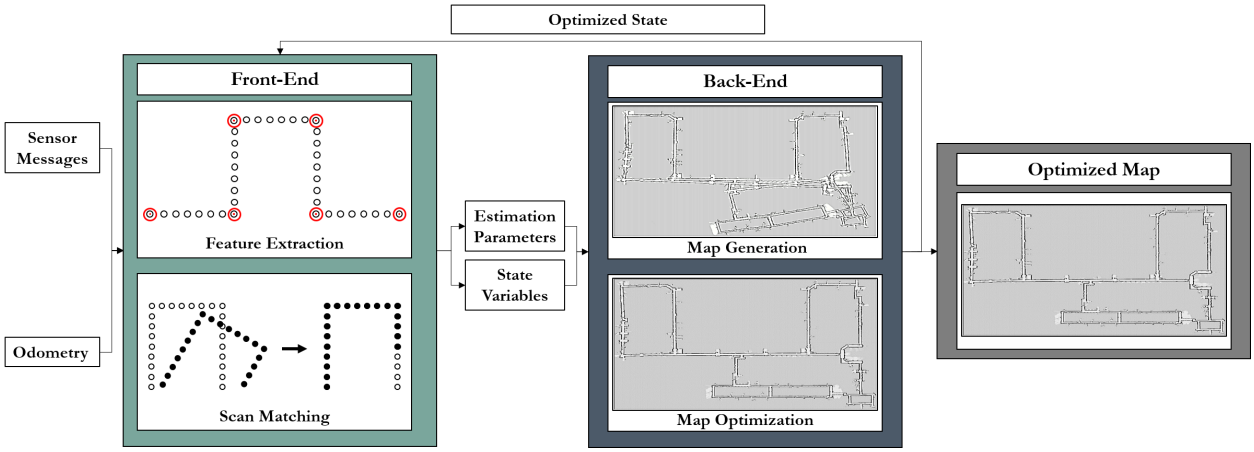


Figure 3.4: Overview of SLAM Framework.

ROS includes several 2-D SLAM packages. These packages are used to build accurate 2-D occupancy grid maps, seen in Figure 3.5. These maps are then used to localize the robot within its environment, as well as to plan and execute appropriate trajectories in order to reach its destination.

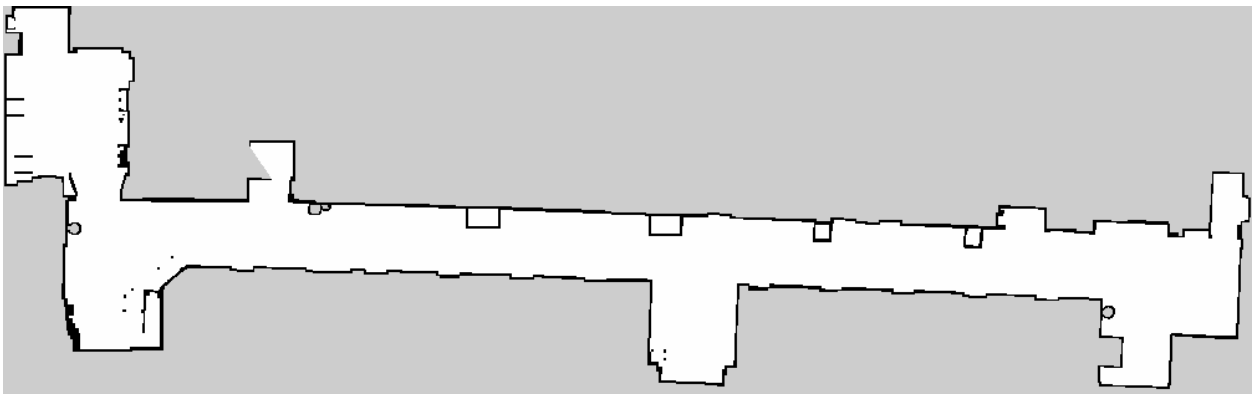
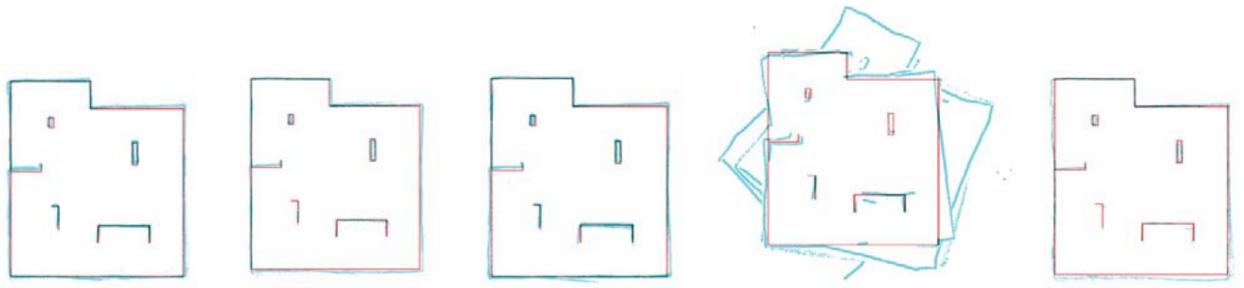


Figure 3.5: Example of 2-D Occupancy Grid Map Produced.

Five of the most common 2-D SLAM ROS packages include HectorSLAM, GMapping, KartoSLAM, CoreSLAM, and LagoSLAM. HectorSLAM [16] is neither a filtering nor an optimization-based smoothing techniques and relies solely on robust scan matching [17]. CoreSLAM [18] and GMapping [19] are filtering SLAM techniques. Both CoreSLAM and GMapping utilize a particle filter. While CoreSLAM employs a very simple particle filter [20], GMapping uses a more complex and efficient Rao-Blackwellized particle filter [21]. Both KartoSLAM [22] and LagoSLAM [23] are optimization-based smoothing SLAM techniques.

However, KartoSLAM, which was developed by SRI robotics, uses a highly-optimized and non-iterative Cholesky matrix decomposition for sparse linear systems, known as Sparse Pose Adjustment (SPA) [15], [24], while LagoSLAM uses LAGO optimizer developed by Carlone et al [25].

Figure 3.6 and Table 3.1 show the aforementioned 2-D SLAM ROS packages real world performance based on testing done by Santos et al [26]. KartoSLAM achieved the smallest error demonstrating the robustness of its sparse pose adjustment (SPA) and that of full SLAM techniques in general. As a result, KartoSLAM was used to generate Figure 3.5 as well as the maps used during testing of the multistage localization approach presented in this paper.



(a) HectorSLAM (b) GMapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

Figure 3.6: Real World Performance Analysis of ROS Available SLAM Algorithms. Adapted from [26].

Table 3.1: Real World Error Estimation for ROS Available SLAM Algorithm. Adapted from [26].

Real World Experiments				
HectorSLAM	GMapping	KartoSLAM	CoreSLAM	LagoSLAM
1.1972	2.1716	1.0318	14.75333	3.0264
0.5094	0.6945	0.3742	7.9463	0.8181
1.0656	1.6354	0.9080	7.5824	2.5236

3.3 Localization and Path Planning Concepts

ROS's Navigation Stack [27] is a collection of packages, which uses odometry and laser scan data, as well as a goal position and orientation in order to output the velocity commands

needed to reach the specified goal. Figure 3.7 shows an overview of how the individual packages work together to achieve this objective. The Map_Server node [28] loads a previously generated two-dimensional grid map. Once the AMCL [29] node receives the map, odometry, and laser scan data, it is able to localize the robot within the provided map, using the Adaptive Monte Carlo Localization technique for which it gets its name. The Move_Base [30] node maintains both global and local planners and costmaps. Information about obstacles in the world are stores in these costmap. The global costmap is used for long-term planning, while the local costmap is used for short-term planning and obstacle avoidance. The global planner computes an optimal path to the goal given the starting state of robot and the global costmap. While the local planner computes shorter trajectories given the current state of the robot and the local costmap. Once a path is developed, Move_Base outputs the necessary velocity commands needed in order to reach the specified destination [27].

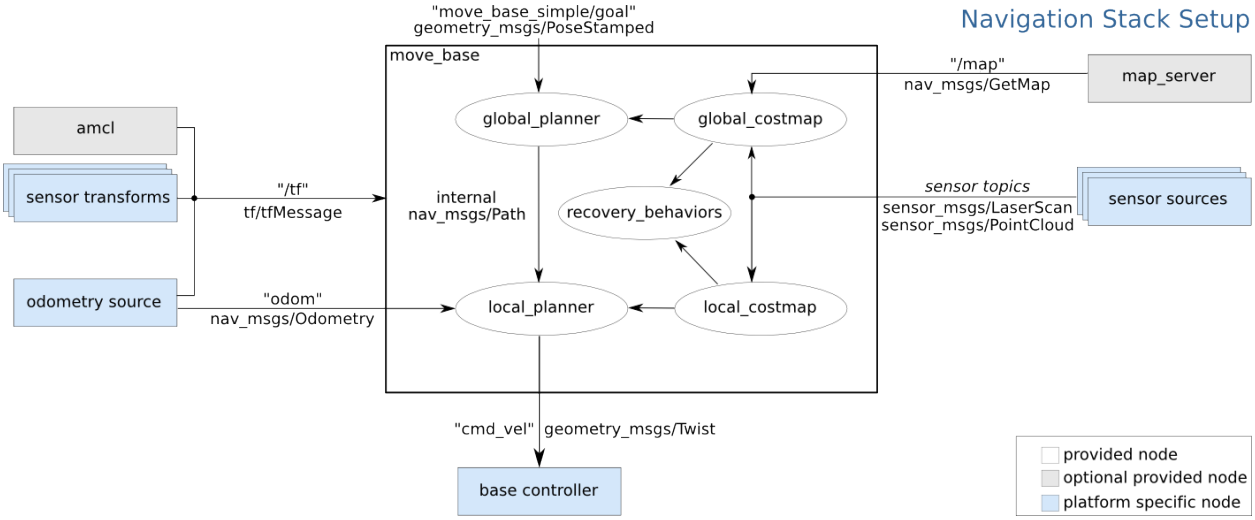


Figure 3.7: ROS Navigation Stack setup [27].

After building a two-dimensional occupancy grid map, shown in Figure 3.5, using SLAM, it becomes crucial to accurately localize the robot within this predefined map; so that, the robot can both plan and execute appropriate trajectories to reach its destination. Localization involves estimating the position and orientation of the robot, known collectively as pose, while the robot moves throughout its environment. One routine localization technique used involves tracking the robot from an initial known starting pose. Through the measurement of wheel rotation and the integration of accelerations provided by an inertial measurement unit (IMU), the distance traveled by the robot from the initial position can be calculated and the robot's pose in the map estimated with some certainty. However, these methods do not account for wheel slippage or measurement error. As a result, the accuracy of the pose estimate will degrade over time. Consequently, a solution which can compensate for

the accumulated odometry error and inaccuracies in the initial starting pose is needed. One accepted solution to this problem is Monte Carlo Localization (MCL), which utilizes a particle filters to keep track of the robot's pose. However, additional options include Kalman Filters and Markov Localization, which employ Gaussian distributions and histograms respectively.

Figure 3.8 depicts MCL using a one-dimensional corridor with a few doors. The robot initially has no information about where it is in this corridor. As a result, the graph of the robot's belief states, which defines the probability of the robot being at a particular position, is drawn from a uniform distribution of discretely sampled positions along the corridor. A measurement update is performed at each time step. A measurement update involves convolving the measurement model, the probability of receiving a specific sensor measurement in the corridor, with the belief states to get an updated belief state. The updated belief state is the same as the pervious belief state; however, the weight of each particle have been updated based on the sensor reading. At step k=1 the robot senses a door; so, the weight of particles at the three door are increased. At the next step, a motion model update is performed. The odometry indicated that the robot moved forward a specified distance d. As a result, the belief state is updated by moving the particles forward that specific distance with noise added to account for the aforementioned odometry errors. It should be noted that the particles at this stage in Figure 3.8 were also resampled, which will be covered in the following paragraphs. The motion model update is followed by a measurement update. The robot again senses a door. As a result, the measurement model is the same as the pervious time step. After convolving the current measurement model with the current belief state, the cumulative probability mass is centered at door two indicating that the robot is likely at this location [31].

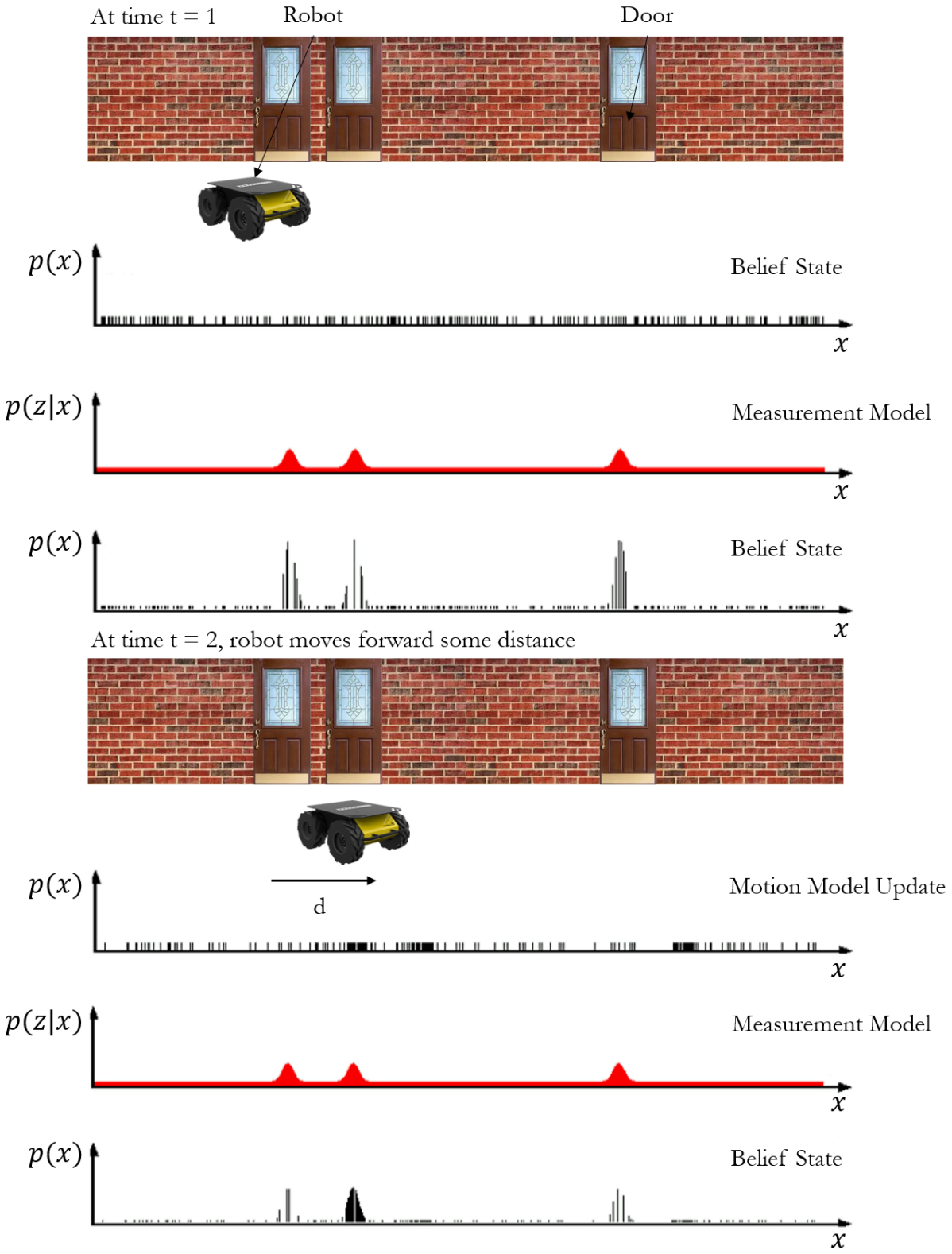


Figure 3.8: One-Dimensional Monte Carlo Localization Example. Adapted from [31].

Two-dimensional MCL follows the same format of a motion model and then measurement update. The initial particles are drawn from the current odometry with added noise. At each step, the particles are updated via the odometry and then corrected via a measurement update. For each particle, the correlation between the two-dimensional occupancy grid map, seen in Figure 3.5, and laser scan is calculated using Equation 3.3, where A is the predefined map, B is the map created by the current laser scan, and $\bar{A}$ and $\bar{B}$ are the mean values of the pixels of both maps respectively. Note that while obstacle pixels are black and have a value of 1, free space pixels are grey and have a value of 0. While m and n are the x and y values of the pixel. The particle (pose) with the highest correlations score is chosen as the pose for the current step. The weight of each particles at step k is found by multiplying the particles weight at step $k-1$ by its normalized correlation score at time step k, as seen in Equation 3.4.

$$s = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}} \tag{3.3}$$

$$W_k \leftarrow W_{k-1} s \tag{3.4}$$

As weights are multiplied over steps, the particles with consecutive small correlation scores are reduced to very small values. As a result, the particles filter eventually has every few effective particles to ensure that good results are produced. Consequently, re-sampling, show in Figure 3.9, is performed when the number of effective particles become too small. This is done by drawing samples close to the particles that have a higher weights, indicated by their size. Thus the new sample have a higher density near the position where the particle with higher weight existed. Each particle after resampling has same weight. As a result, the particle filter begins from scratch.

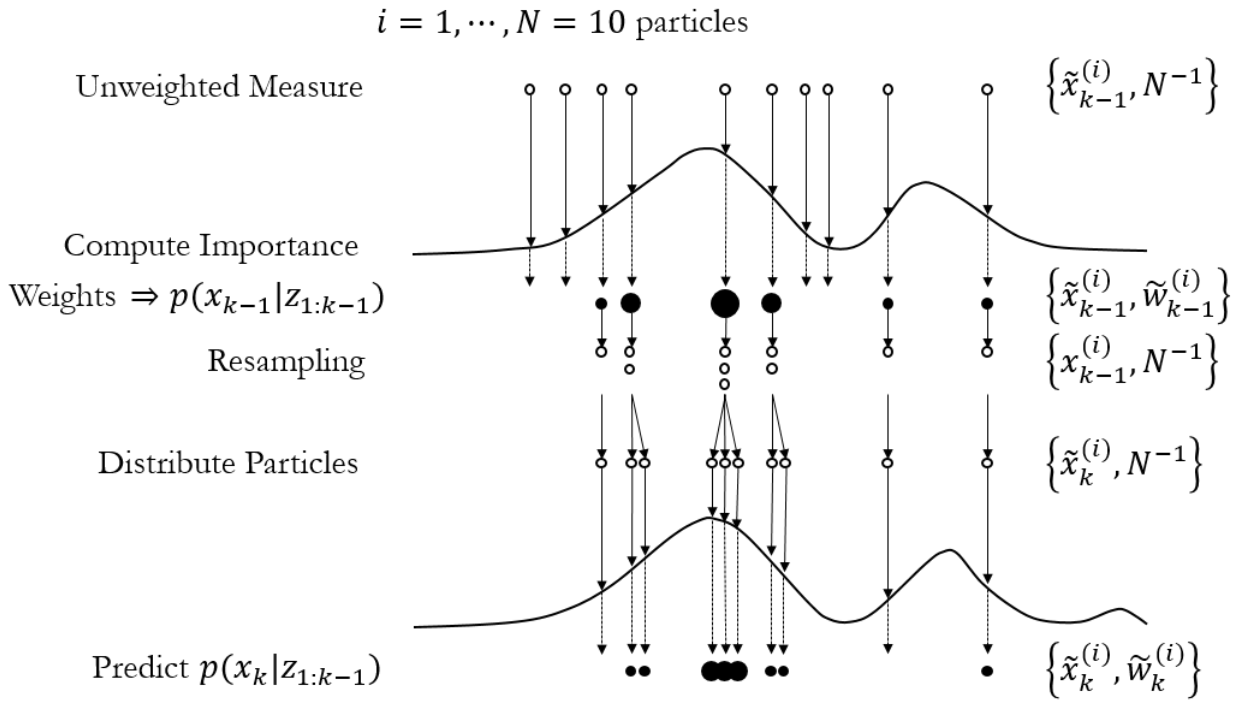


Figure 3.9: Particle Filter Resampling Example

Due to the computational complexity inherent in iteratively having to calculate the correlation score for each particle, an optimization technique known as *KLD-sampling* is used. *KLD-sampling*, derived from *Kullback-Leibler divergence*, is a technique that determine the number of particles needed such that the error between the sample and true posterior is less than ϵ [31]. KLD-sampling basically control the number of particles based on the difference in odometry and particle base location.

Figure 3.10 shows how KLD-sampling effectively works. Initially when the position is unknown, the particle cloud is large due to the uncertainty in the position and orientation of the robot. However, as the robot moves, the particle converges and the particle cloud size reduces as KLD-sampling removes the redundant particles and improves computational performance.

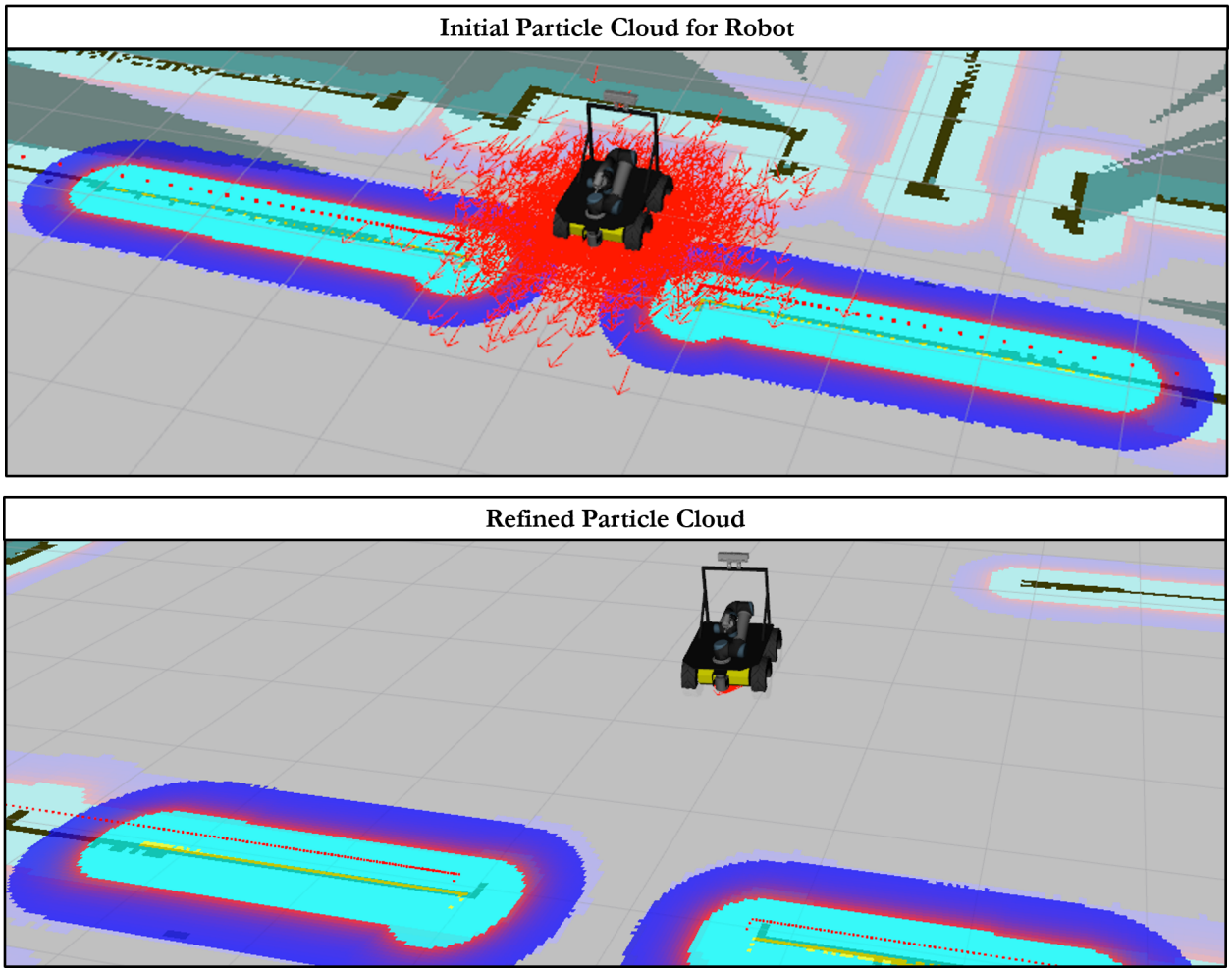


Figure 3.10: Visual of AMCL in RVIZ

After calculating the local and global costmaps, as well as the location of the robot within the given map, the robot must now both plan and execute appropriate trajectories to its destination. Two frequent techniques used are the Dynamic Window Approach (DWA) and Trajectory Rollout. Both sample the space of feasible controls. For a differential drive robot, such as Clearpath Robotics' Husky and Fetch Robotics' Fetch, this controls space is 2D and consists of translations and rotational velocities, $\dot{x}, \dot{\theta}$, which are limited by the robot's capabilities. Each sampled velocity is forward simulated from the robot's current position for a short period of time in order to generate simulated trajectories as shown in Figure 3.11. These simulated trajectories are then scored using the cost function in Equation 4.1,

$$C(k) = \alpha Obs + \beta Gdist + \gamma Pdist + \delta \frac{1}{\dot{x}^2} \tag{3.5}$$

where *Obs* is the sum of grid cell cost through which the trajectory passes (taking account of the robot's actual footprint in the grid); *Gdist* and *Pdist* are the estimated shortest distance from the endpoint of the trajectory to the goal and the optimal path, respectively; and $\dot{x}$ is the translation component of the velocity command that produces the trajectory.

The simulated trajectory that minimizes this cost function is chosen. As a result, chosen trajectories tend to keep obstacles at a distance, proceed towards the goal, remain near the optimal path, as well as have higher velocities [32]. DWA and Trajectory Rollout differ in that Trajectory Rollout samples achievable velocities over the entire forward simulation, while DWA sample only from achievable velocities for just one simulation step [33].

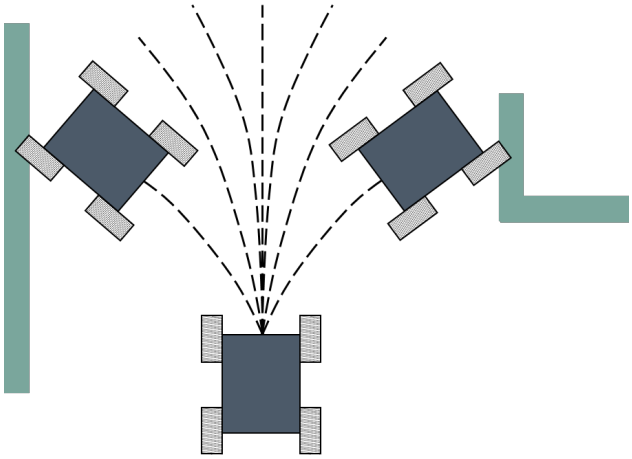


Figure 3.11: Trajectory Rollout Path Planning Framework.

3.4 Manipulation Concepts

ROS's MoveIt! [34] is a collection of packages for mobile manipulation, which incorporate the latest advances in motion planning, manipulation, 3-D perception, kinematics, control, and navigation. Figure 3.12 shows Moveit's overall system architecture. The move_group node integrates all individual packages together in order to provide the user a set of ROS actions and services. The user interface allows the user to interact with the move_group node through the ROS interface by using C++, Python, or RVIZ [35], which is ROS's graphical user interface and 3-D visualization tool. ROS's Parameter Server [36] provides the move_group node the robot's URDF and Semantic Robot Description Format (SRDF) files, as well as Moveit! specific configuration files. The SRDF is used to represent information about the robot that is not included in the URDF file, such as a set of links or joints, known collectively

as a group, that make up the manipulator, predefined group states, and a lists of links between which collision checking should be disabled [37]. Moveit! configuration files include ones that set joint limits, as well as kinematics, motion planning and perception parameters. The SRDF, as well as necessary Moveit! configuration files are setup through Moveit!'s Setup Assistant, which is Moveit!'s graphical user interface for configuring any robot for use with Moveit!. The robot interface allows Moveit! to send command to and receive feedback from the robot. Moveit!'s Planning Scene Monitor, which monitors information from the robot's sensors, is used to maintain a planning scene, which represent the world around the robot and stores the state of the robot itself. This information is used by Moveit!'s Flexible Collision Library, as well as the motion and kinematic planning plugins to plan and execute obstacle free paths for the manipulator [38].

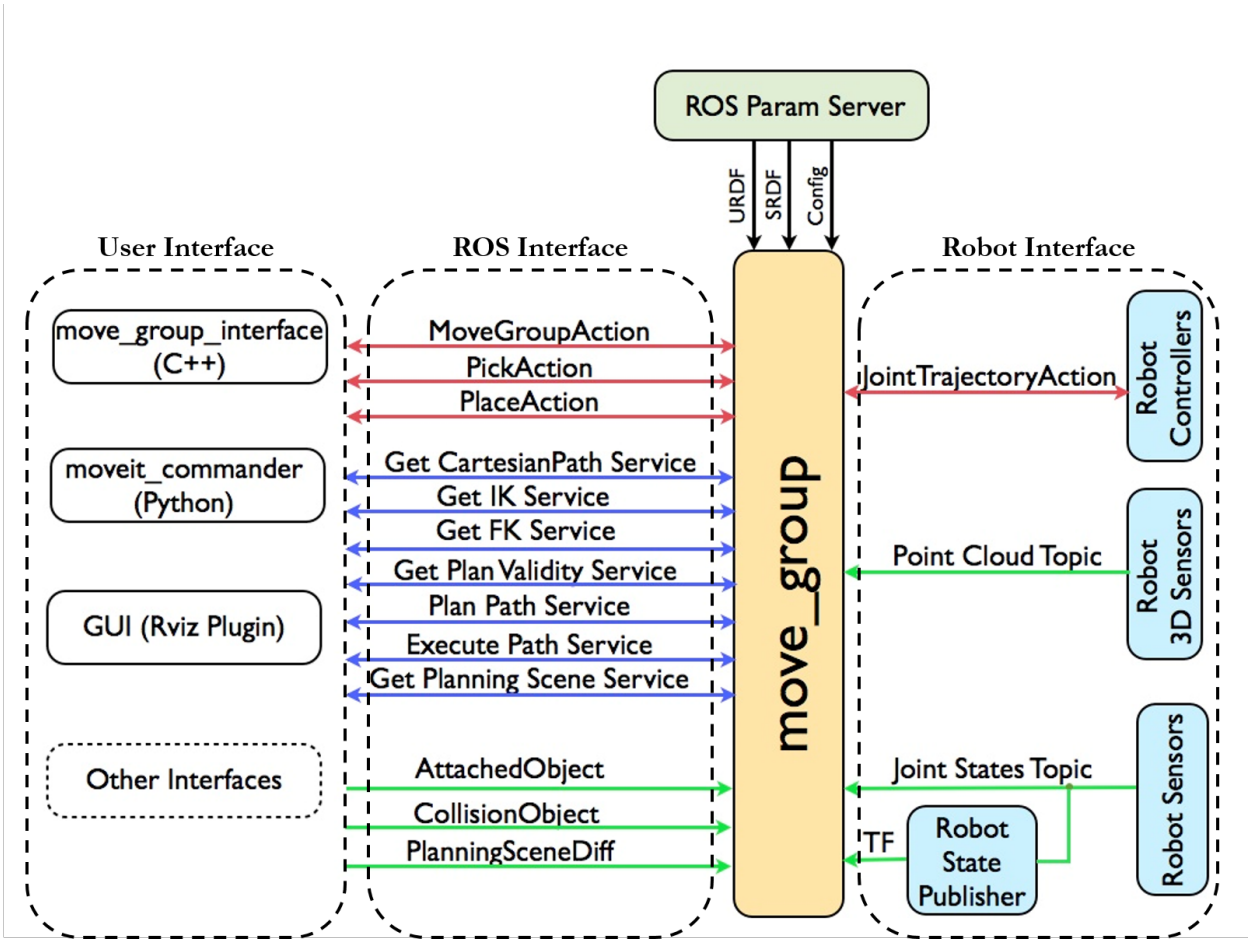


Figure 3.12: Moveit!'s System Architecture. Adapted from [38]

Moveit! allows the user to plan trajectories either in joint space, forward kinematic (FK), or Cartesian space, inverse kinematics (IK). Forward kinematics uses the manipulator's

kinematic equations to compute the positon of the end-effector given specific predetermined joint angles [39]. Whereas, inverse kinematics uses the manipulator's kinematic equation to determine the joint angles necessary for the end-effector to arrive at the desired position [40]. Figure 3.13 illustrated the difference between forward and inverse kinematics. In most cases, including the multistage localization approach presented in this paper, the target joint angles are not known ahead of time. As a result, a target pose is specified for the manipulator's end-effector in Cartesian space and Moveit!'s IK solver will determine the appropriate joint angles to reach the desired pose [41].

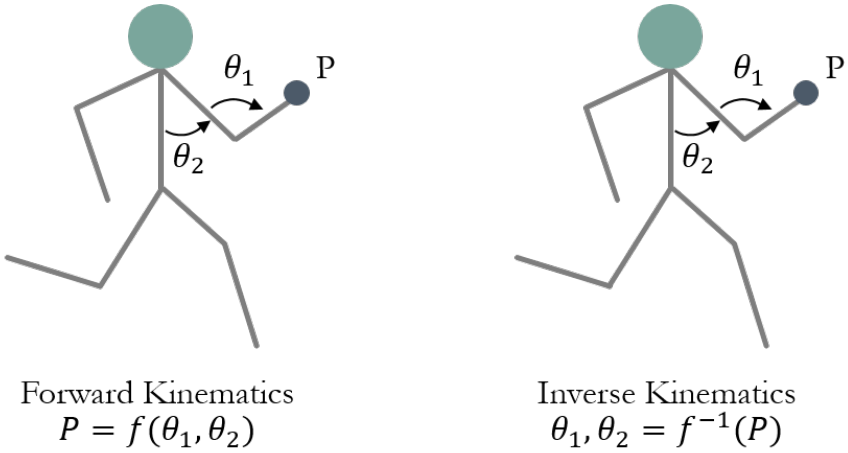


Figure 3.13: Forward and Inverse Kinematics Example

There are two main approaches for solving IK problems: numerical and analytical (close form) solutions. Due to the recursive nature of numerical solvers, they tend to be slower than that of analytical solvers and are prone to get trapped in local minima [42]. However, analytical solvers require that a separate kinematic solver be created for each manipulator and manipulator configuration. In order to ensure cross platform functionality with minimum overhead, only available numerical kinematic solvers were examined. However, OpenRAVE (Open Robotics Automation Virtual Environment) [43], which is software for simulating and deploying planning algorithm, does provided a tool called IKFast [44], which allows users to generate custom analytical kinematic solver, which can produce solution on the order of approximately four microseconds. Table 3.2 shows three common ROS Moveit! numerical kinematic solvers implementations, Orocos KDL [45], as well as the TRACLab's [46] KDL-RR and TRAC_IK [47]. Orocos Kinematics and Dynamics Library (KDL) is a joint-limit-constrained pseudoinverse Jacobian solver, whose convergence algorithms are based on Newton's method. However, KDL suffers from the following issues, which results in the high failure rates and slow average speed seen in Table 3.2.

1. Frequent convergence failures due to manipulators with joint limits

2. No action is taken when the search gets trapped in a local minima
3. Lack of tolerance support and/or utilization in the solver itself

The TRACLab's KDL-RR kinematic solver implementation attempts to solve issues two and three by altering Orocos's KDL implementation to detect and "untick" the iterative convergence algorithms, as well as to loop for a maximum time versus a maximum number of iteration. KDL-RR is able to detect local minima by monitoring when the difference between joint angles across iterations become approximately zero. When this situation occurs, random seed angles are introduced, which "unstick" the convergence algorithms. In addition, KDL-RR loops for maximum time versus maximum iteration due to the fact that the maximum number of iteration is an arbitrary number, while maximum time can be computed based on the size and complexity of the each manipulator. In addition, using maximum time makes comparing implementations easier. Table 3.2 shows that KDL-RR had a dramatic increase in solve rate percentage, as well as a modest decrease in runtime compared to KDL. However, KDL-RR's doesn't solve issue one. Consequently, the failure rate is still unacceptably high. TRACLab TRAC-IK kinematic solver attempts to solve issue one by using two separate IK solvers concurrently. It uses KLD-RR, as well as a sequential quadratic programming (SQP) IK implementation, which is an iterative algorithm for nonlinear optimization. The SQP implementation, which incorporates the same local minimum detection, is able to deal with issue one. However, SQP can have a much longer solve rate than KDL or KDL-RR. As a result, TRAC-IK implements both of these IK method concurrently and waits for either to converge. As a result, the solve rate range between 99.1% and 99.9 % and the average convergence times are well below one millisecond [48].

Table 3.2: Comparison Between ROS Available Moveit! Inverse Kinematic Plugins. Adapted from [48].

Kinematics Chain		IK Technique					
Robot	DOFs	Orocos' KDL		KDL-RR		TRAC-IK	
		Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)
Atlas 2013 Arm	6	75.54	1.35	97.13	0.39	99.97	0.33
Atlas 2015 Arm	7	75.71	1.50	93.13	0.81	99.18	0.48
Baxter Arm	7	61.07	2.21	89.52	1.02	99.17	0.60
Denso VS-068	6	27.92	3.69	98.13	0.42	99.78	0.38
Fanuc M-430iA/2F	5	21.07	3.99	88.34	0.92	99.16	0.58
Fetch Arm	7	92.49	0.73	93.82	0.72	99.96	0.44
Jaco2	6	26.23	3.79	97.66	0.58	99.51	0.58
LBR IIWA 14 R820	7	37.71	3.37	94.02	0.73	99.63	0.56
KUKA LWR 4+	7	67.80	1.88	95.40	0.62	99.95	0.38
PR2 Arm	7	83.14	1.37	86.96	1.27	99.84	0.59
NASA Robonaut2 'Grasping Leg'	7	61.27	2.29	87.57	1.10	99.31	0.67
NASA Robonaut2 'Leg + Waist + Arm'	15	97.99	0.80	98.00	0.84	99.86	0.79
NASA Robonaut2 Arm	7	86.28	1.02	94.26	0.73	99.25	0.50
NASA Robosimian Arm	7	61.74	2.44	99.87	0.36	99.93	0.44
TRACLabs Modular Arm	7	79.11	1.35	95.12	0.63	99.80	0.53
UR10	6	36.16	3.29	88.05	0.82	99.47	0.49
UR5	6	35.88	3.30	88.69	0.78	99.55	0.42
NASA Valkyrie Arm	7	45.18	3.01	90.05	1.29	99.63	0.61

3.5 Task Execution Concepts

It is relatively straightforward to program a robot to execute an individual action. However, programming a fully autonomous robot that will be expected to select which individual action it will perform depending upon the task at hand and the current conditions isn't so straight forward. The task execution system must be able to rank tasks by priority, break down tasks into subtasks, execute tasks in parallel, monitor conditions and react according, as well as pause and resume tasks at a later if necessary [41]. A commonly used construct to achieve the aforementioned requirements is a hierarchical state machine.

Finite state machines are used to model control and sequences in a system [49]. In a state machine, the robot occupies one state, which has set behaviors or tasks associated with it. As long as the robot remains in that states, it will continue to carry out the same behavior or tasks. States are connected together by transitions. When certain predefined conditions are met, a transition is triggered and the system changes from its current state to the target state. State machines are a powerful tool; however, it can be difficult to express some behaviors, such as "alarm behaviors". Image an autonomous industrial mobile manipulator (AIMM) that is responsible for drilling specific patterns on different sections of plane wings. This can be easily implemented using a normal state machine. The AIMM will need to navigate to the specific starting location of each wing, determine the appropriate drill locations based

on the associated pattern, and finally carry out the individual drilling operations. This can be easily implemented using a normal state machine. Unfortunately, the AIMM's battery does not provide power indefinitely. When the AIMM's power level drops to a certain level, it will need to stop and navigate to the nearest charging location to be recharged, regardless of what it is doing at the time. When it is fully recharged, it will need to pick up exactly where it left off. The recharging periods is an alarm mechanics: something that interrupts normal behavior to respond to something important. Representing this in a state machine leads to a doubling the number of states. With only one level of "alarm behaviors', this is no problem; however, if we add levels of "alarm behaviors", the number of states will increase exponentially. So, rather than combining all the logic into a single state machine, the logic can be separated it into several. Each alarm mechanism should have its own state machine, along with the original behavior. These individual state machines are arrange into a hierarchy, so the next state machine down is only considered when the higher level state machine is not responding to its alarm. The nesting of states machines inside another makes what is called a hierarchical state machine [50].

ROS's SMACH [51] package includes a standalone python library for programming hierarchical state machines, as well as a ROS wrapper for integrating the library with ROS topics, services, and actions. Figure 3.2 shows a graphical view of a hierarchical state machine set up using SMACH. This hierarchical state machine simulates a robot performing tasks such as drilling and sealing in a manufacturing environment while having the option to change its tool or recharge if necessary.

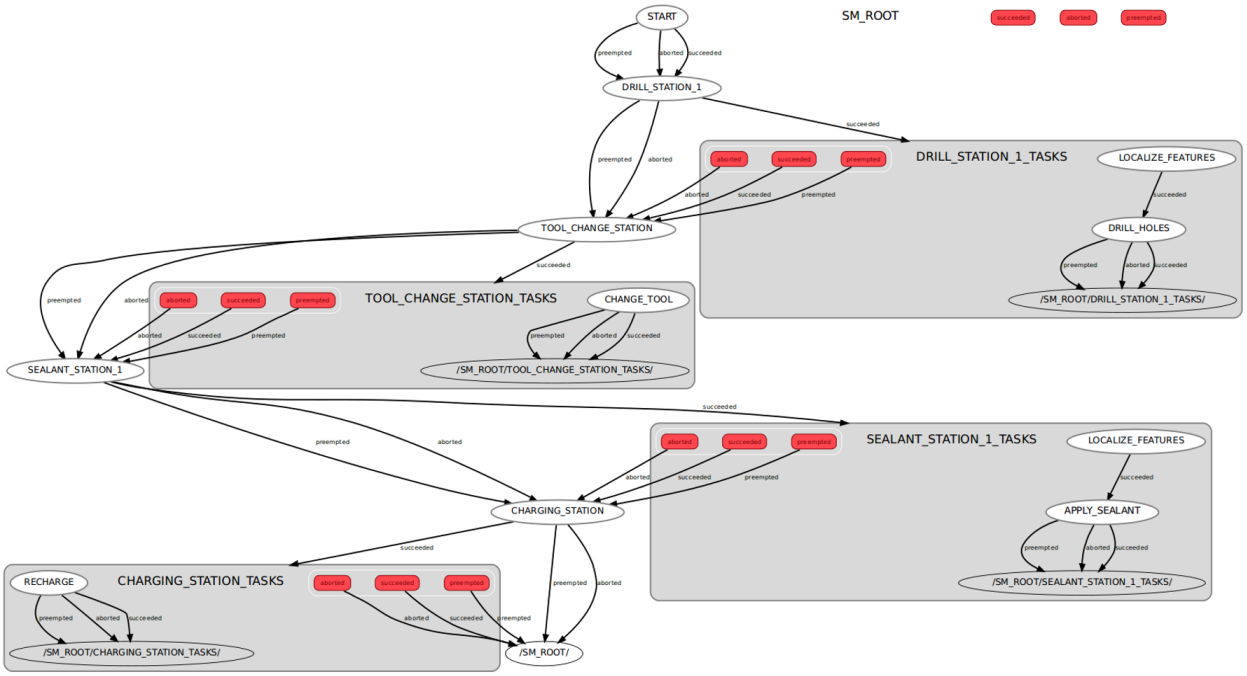


Figure 3.14: Graphical View of State Machine using SMACH

3.6 Camera Concepts

An image is the optical representation of an object illuminated by a radiating source. Figure 3.15 shows a simplified model of how photometric images are formed.

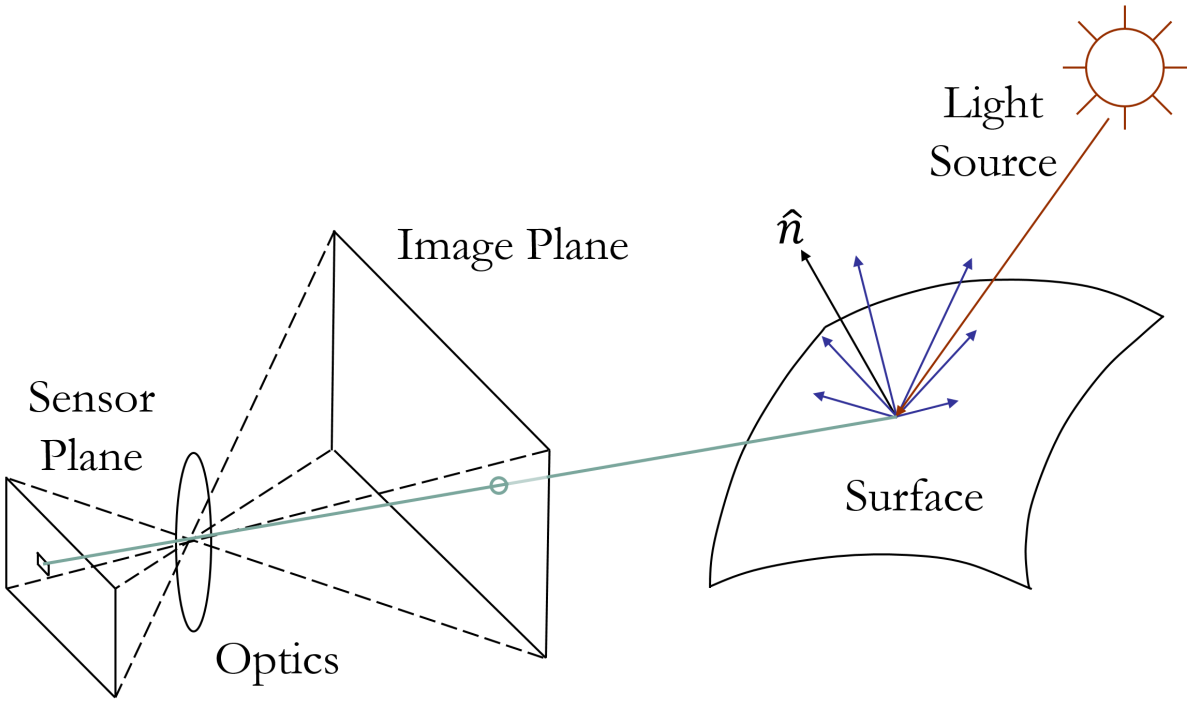


Figure 3.15: Photmetric Image Formation. Adapted from [52].

Visible light is emitted by one or more sources, which is then reflected off an object's surface. This reflected light, is the optical image which is the input of a digital image formation systems [53], such as a digital camera, depicted in Figure 3.16.

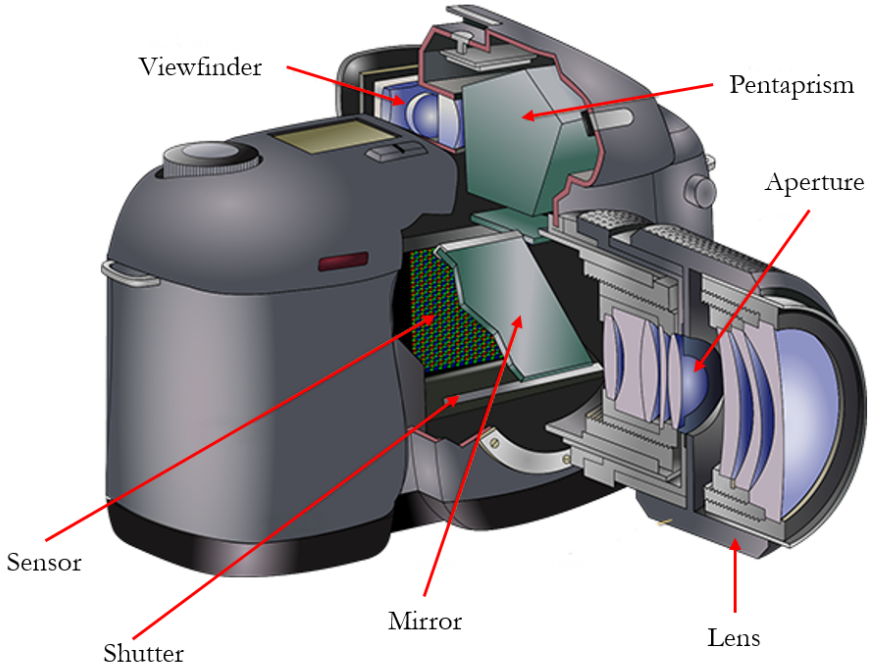


Figure 3.16: Digital Camera Diagram. Adapted from [54].

The light passes through a convex lens which focuses all of the rays through the optical center point of the lens and onto a silicon sensor. This sensor, also referred to as the sensor plane, is essentially a 2-D array, which consists of thousands to millions of solar cells that convert the light into electrons. The accumulated charge of each cell is transmitted to an onboard computer which digitizes this information; so that, a computer can render the digital image of the world [55]. Since the sensor plane is vertically rotated 180°, due to the principle of the pinhole camera model [56], the image plane is introduced for mathematically convenience. The image plane, also known as the focal plane, is a virtual plane that is located in front of the optical center of the camera lens and is equivalent to the sensor plane, in that it represents a projection of the world that is mapped onto a plane. In addition, it accounts for the rotation of the image in the sensor plane.

In order for the multistage localization technique presented in this paper to achieve the desired accuracy, a digital RGB camera with the appropriate sensor size, resolution, focal length, frame rate, and shutter type must be chosen. The focal length, which is the distance between the optical center point of the camera lens to the image plane [57], and sensor size of the RGB camera are used to determine both the angular horizontal and vertical field of view (FOV), which can be calculated using Equations 3.6 and 3.7 respectively,

$$HFOV_{angular} = 2\tan^{-1}(\frac{w_s}{2f}) \tag{3.6}$$

$$VFOV_{angular} = 2\tan^{-1}(\frac{h_s}{2f}) \tag{3.7}$$

where w_s and h_s are the weight and heights of the sensor, and f is the focal length. These angular FOVs can be converted from angular to distance measurements, using Equations 3.8 and 3.9,

$$HFOV_{distance} = 2d\tan(\frac{HFOV_{angular}}{2}) \tag{3.8}$$

$$VFOV_{distance} = 2d\tan(\frac{VFOV_{angular}}{2}) \tag{3.9}$$

where d is distance from camera to work surface. In addition, Equations 3.10 and 3.11, where r_h and r_v is horizontal and vertical resolution respectively, can be used to calculate the horizontal and vertical ground sample distance (GSD). GSD is the distance between two consecutive pixels centers. Whereas, spatial resolution is the area of each pixels, which is a function of the horizontal and vertical GSD [58]. The ground sample distance is the highest accuracy a localization system can achieve. Figure 3.17 shows the effect that GSD can have on the detail of an image and consequently the accuracy of a localization system.

$$HGSD = \frac{w_s d}{f * r_h} \tag{3.10}$$

$$VGSD = \frac{h_s d}{f * r_v} \tag{3.11}$$

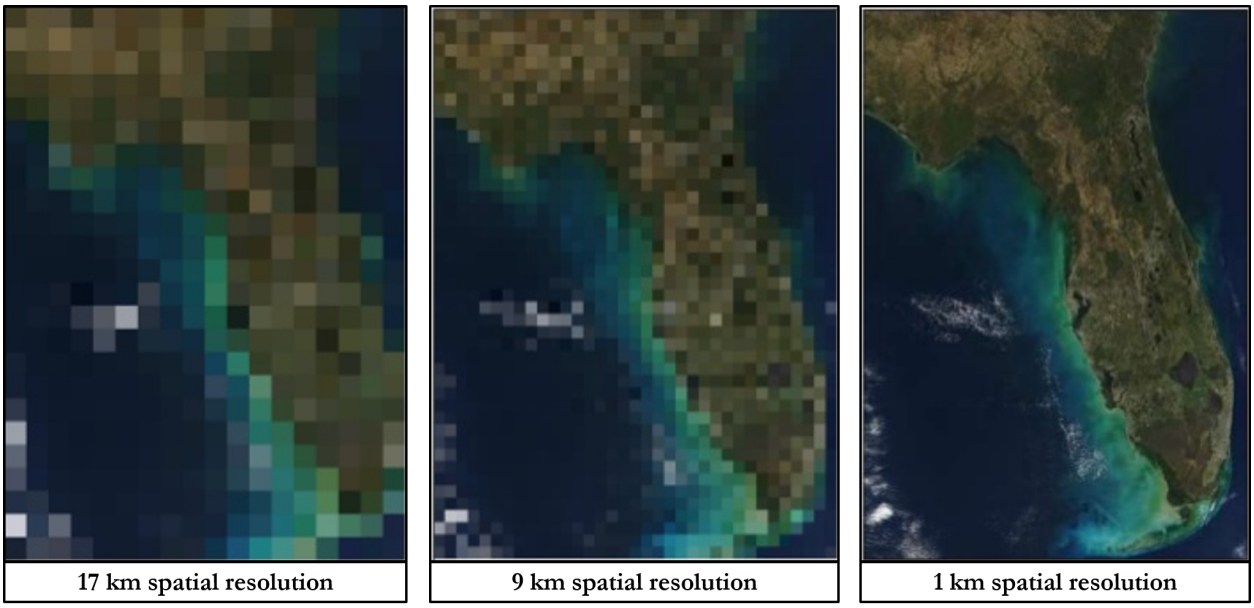


Figure 3.17: Ground Sample Distance Effects on Image Quality. Adapted from [59].

Equations 3.8 and 3.11, all depend on d, the distance from the work surface. Knowing the desired accuracy, as well as horizontal and vertical FOV of a certain application, an appropriate d can be calculated by rearranging the equations above.

In order for the localization technique to be able to update in real time, even as the systems moves, two additional camera features much be considered, frame rate and shutter type. An appropriate frame rate is necessary to ensure an adequate image overlap; so that, features points can be tracked using optical flow, which will be discussed in Section 3.7.7, instead of having to redetect the feature each frame. The required frame rate can be calculated using Equation 3.12,

$$FR = \frac{v}{FOV_{distance}(1-o)} \tag{3.12}$$

where v is velocity and o is percent overlap. However, due to the fact that while localizing the system may move in both the horizontal and vertical planes, Equation 3.13, where v_h and v_v are horizontal and vertical velocity respectively and o_h is horizontal percent overlap, while o_v is vertical percent overlap, should be used to ensure an adequate frame rate in both planes [60].

$$FR = \sqrt{\frac{v_h}{HFOV_{distance}(1-o_h)} + \frac{v_v}{VFOV_{distance}(1-o_v)}} \tag{3.13}$$

There are two types of camera shutters, rolling shutter (RS), in which the horizontal rows of the sensor array are scanned at different time, and global shutter (GS), in which all pixel are exposed at the same time. Due to the RS not exposing all pixels simultaneously, they are susceptible to motion blur [61] as seen in Figure 3.18. Consequently, in order to ensure features are not lost between frames as the system move, a digital RGB camera with a GS is needed.

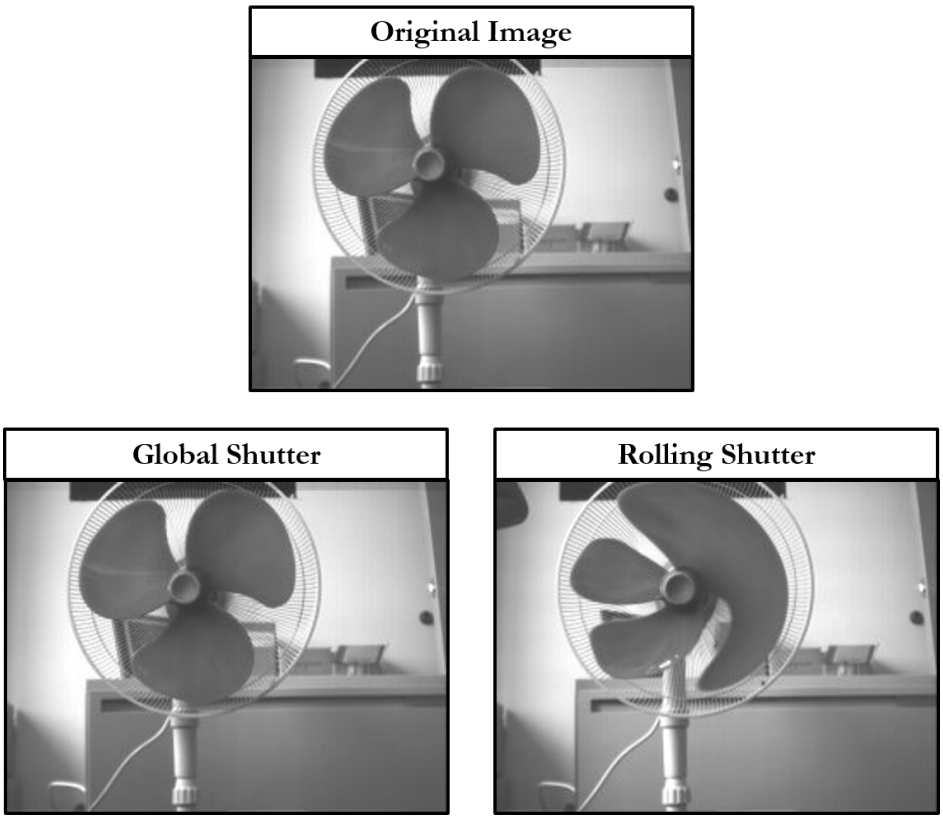


Figure 3.18: Shutter Type Effects on Image Quality. Adapted from [62].

3.7 Computer Vision Concepts

The purpose of this subsection is to briefly explain all the computer vision concepts needed to understand the multistage localization approach presented in the subsequent chapters. The following subsections will explain the basic concepts and rational behind the use of certain color spaces, filters, and binary operations. In addition, Canny Edge Detection, Hough Circle Detection, Good Features to Track, and Optical Flow, as well as Augmented Reality tag detection and pose estimation will be expounded upon.

3.7.1 Color Spaces

While the RGB color space is the primary color space used to describe spectral content of color signals, a variety of other representations have been developed [52]. Each color space has unique advantages and disadvantages that must be considered before using a specific color space for a given application. The two color spaces used in the multistage localization approach presented in this paper are the RGB and the HSV color spaces, depicted in Figure 3.19. The RGB color space is an additive color space based on the RGB color model, in which red, green, and blue light is added together to produce any color that is the triangle defined by those three primary colors [63], [64]. The HSV color space, which was developed by Alvy Ray Smith in 1978, is a cylindrical-coordinate representations of points in the RGB color model [65]. In some situations, such as color picking, the HSV model is more intuitive, as it mirror traditional color mixing methods [66]. HSV stands for hue, saturation, and value. Hue represent angular position on the color wheel, where red starts at 0°, green at 120°, and blue at 240°. Saturation, or distance from the center axis, indicates the amount of grey in a color, where 0 is the color grey and 1 is the primary color. Value, or distance along the center axis, represent the brightness of a color, where 0 is black and 1 is white [65].

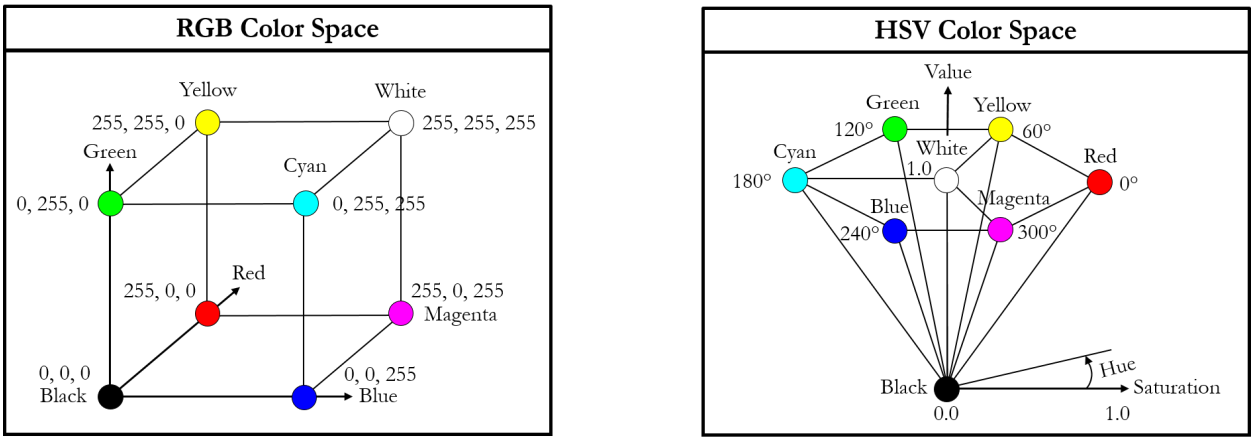


Figure 3.19: RGB and HSV color space models

3.7.2 Linear and Non-Linear Filters

Images are often corrupted by random variations in intensity, illumination, as well poor contrast that must be dealt with in the early stages of image processing [67] before higher level computer vision techniques, such as feature detection and tracking, can be reliably used. Random variations in intensity of an image is called noise. Common types of image noise include salt and pepper, impulse, and Gaussian noise. While salt and pepper noise is the random occurrence of both black and white pixels, impulse noise is merely the random

occurrence of white pixels. Gaussian noise on the other hand contains variation in intensities drawn from a Gaussian distribution, which can be used to model many kinds of sensor noise [68]. In most cases, convolving an image with a linear low-pass filter, a filter which replaces each pixel value with the weighted sum of all pixel values in its neighborhood, or a spatially invariant non-linear filter, which does not use a weighted sum but rather preforms the same operation at each pixel, is able to remove these types of noise. Figure 3.20 shows that a Gaussian filter is apt at removing noise drawn from a Gaussian distribution, such as white noise. While a median filter, which replaces each pixel with the median value in its neighborhood, is suitable for removing salt and pepper, as well as impulse noise. In addition to removing noise, filters can be used to enhance an image, such as sharpen it as seen in Figure 3.20, extract information, like edges and texture, as well as detect patterns through template matching [68].

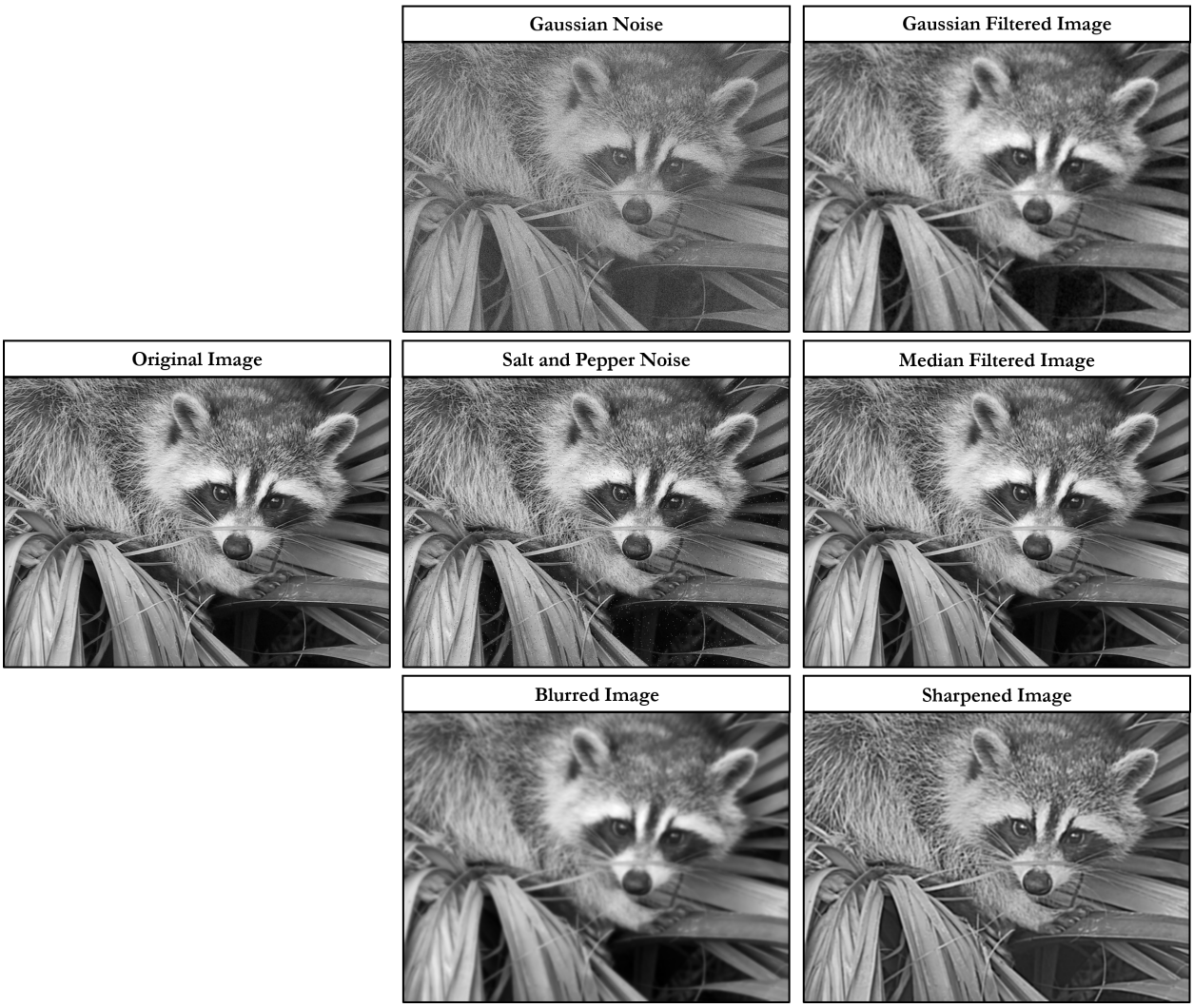


Figure 3.20: Effects of common filters.

3.7.3 Morphological Operations

In addition, to enhancing images, non-linear filters are also used extensively to process binary images [52]. A binary image is an image with only two possible values for each pixel, typically black, background pixels with a value of 0, and white, foreground pixels with a value of 1. Binary images often result after operations such as segmentation, thresholding, and dithering [69]. The most common binary image operations are called morphological operations, since they change the shape of the underlying binary objects [52]. Whether or not a given foreground or background pixel value changes depend on the image, as well as the morphological operations and the structuring element chosen. The structuring element

is a rectangular array of pixels, which contain values of either 1 or 0. When the structuring element is centered on the pixel under consideration, that pixel's neighborhood is determined by the pixels in the structuring element that have a value of 1 [70]. The two fundamental morphological operations are dilation, which thickens foreground pixels, and erosion, which thins foreground pixels. These two operations can be applied in different combinations in order to obtain more sophisticated operations, such as closing, dilation then erosion, and opening, erosion then dilation [71]. The morphological operations, depicted in Figure 3.21, are typically used to clean up binary images; so that, higher level computer vision techniques can be used robustly [52].

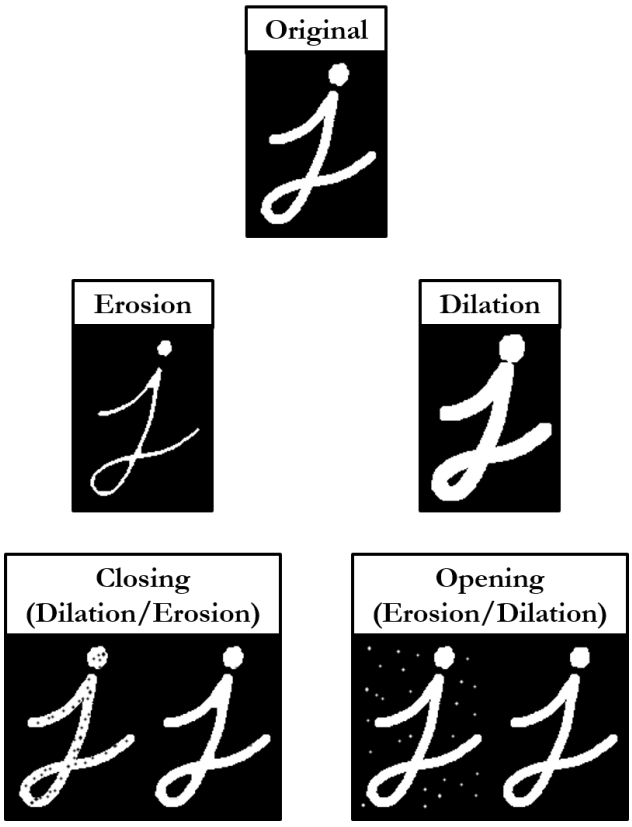


Figure 3.21: Effects of common binary operations.

3.7.4 Canny Edge Detection

Insert Text Explaining the Basic Intuition Behind Canny Edge Detection.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{3.14}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{3.15}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.16}$$

$$\theta = \tan^{-1}(\frac{G_y}{G_x}) \tag{3.17}$$

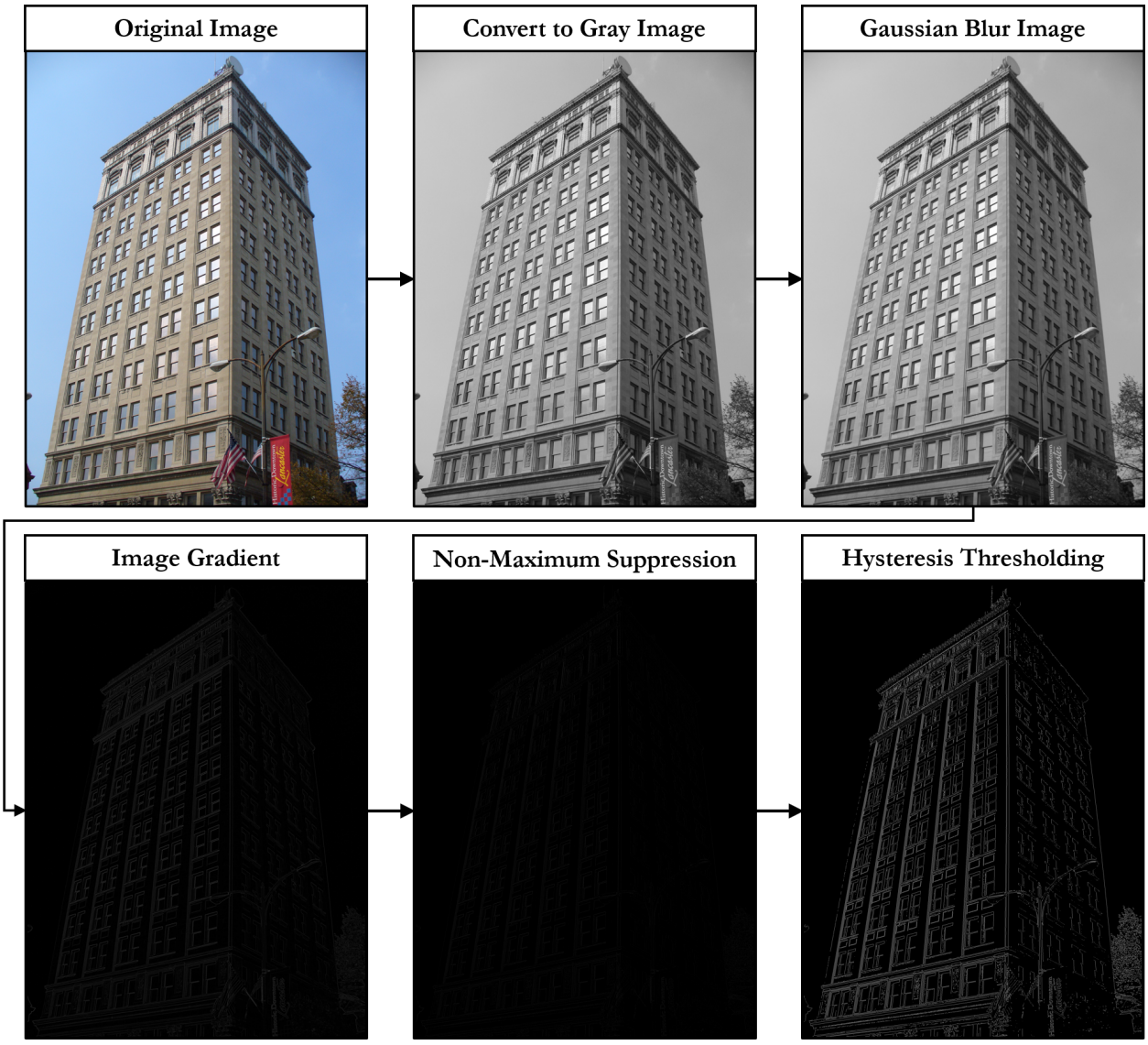


Figure 3.22: Summary of Canny Edge Detection Framework.

3.7.5 Hough Circle Detection

Insert Text Explaining The Basic Preprocessing Operation Performed by Hough Circle Transform In Order to Get Edges.

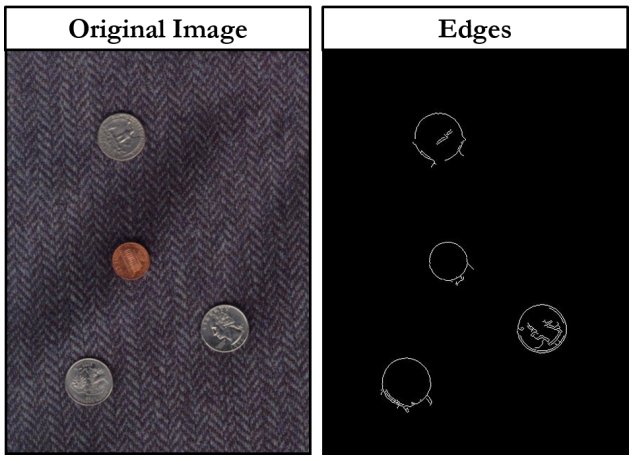


Figure 3.23: Summary of preprocessing operation performed by Hough Circle detector.

Insert Text Explaining the Basic Intuition Behind Hough Circle with Know Radius.

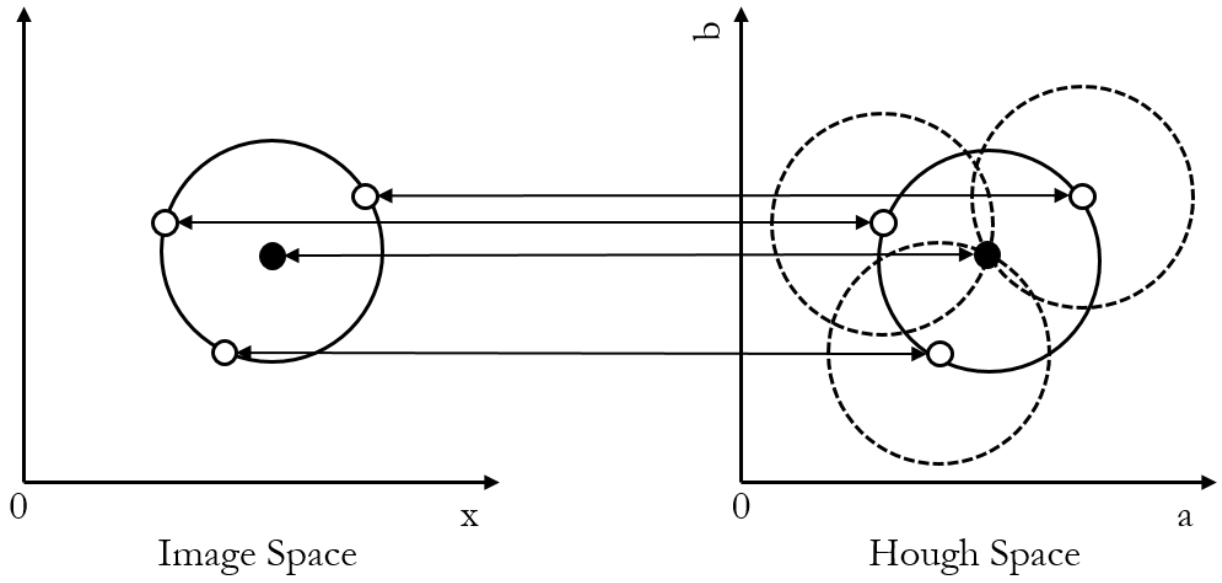


Figure 3.24: Summary of Hough Circle detector with known radius.

Insert Text Expanding Basic Intuition Behind Hough Circle with Unknown Radius.

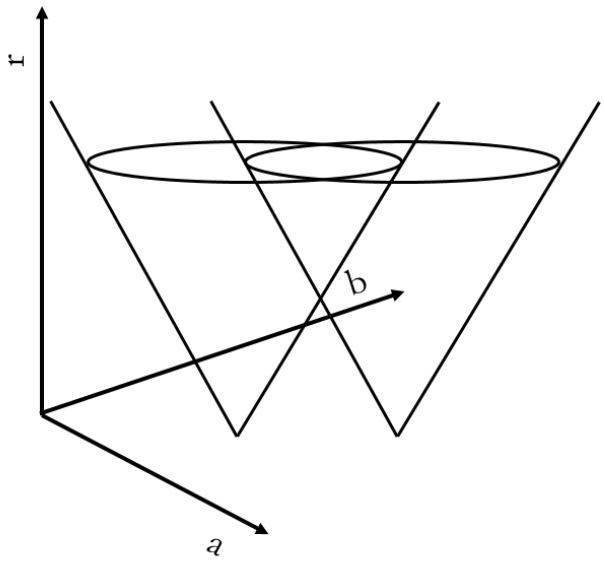


Figure 3.25: Summary of Hough Circle detector with unknown radius.

3.7.6 Good Feature To Track

Insert Text Explaining the Basic Intuition Behind Good Feature.

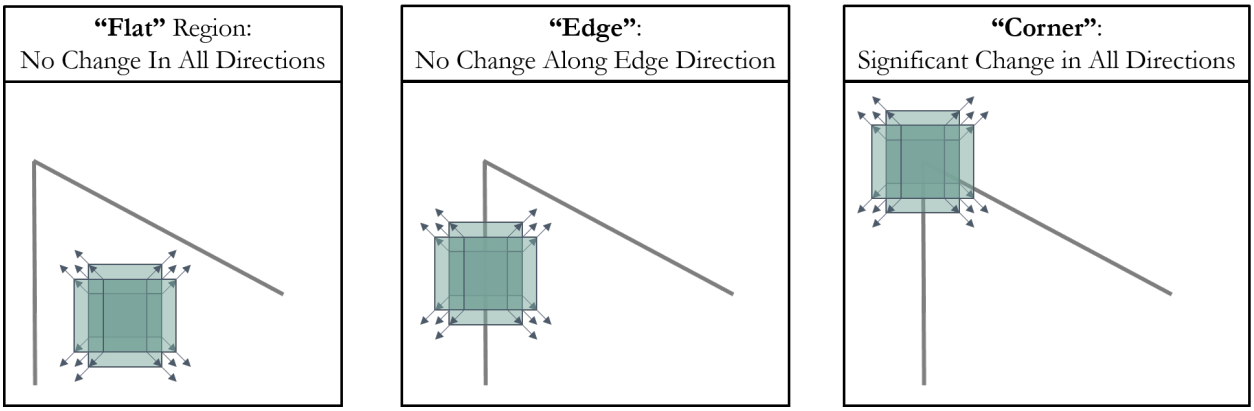


Figure 3.26: Summary of Good Feature detector.

Insert Text Explaining an Image Gradient/Derivative and Why We Look for Them.

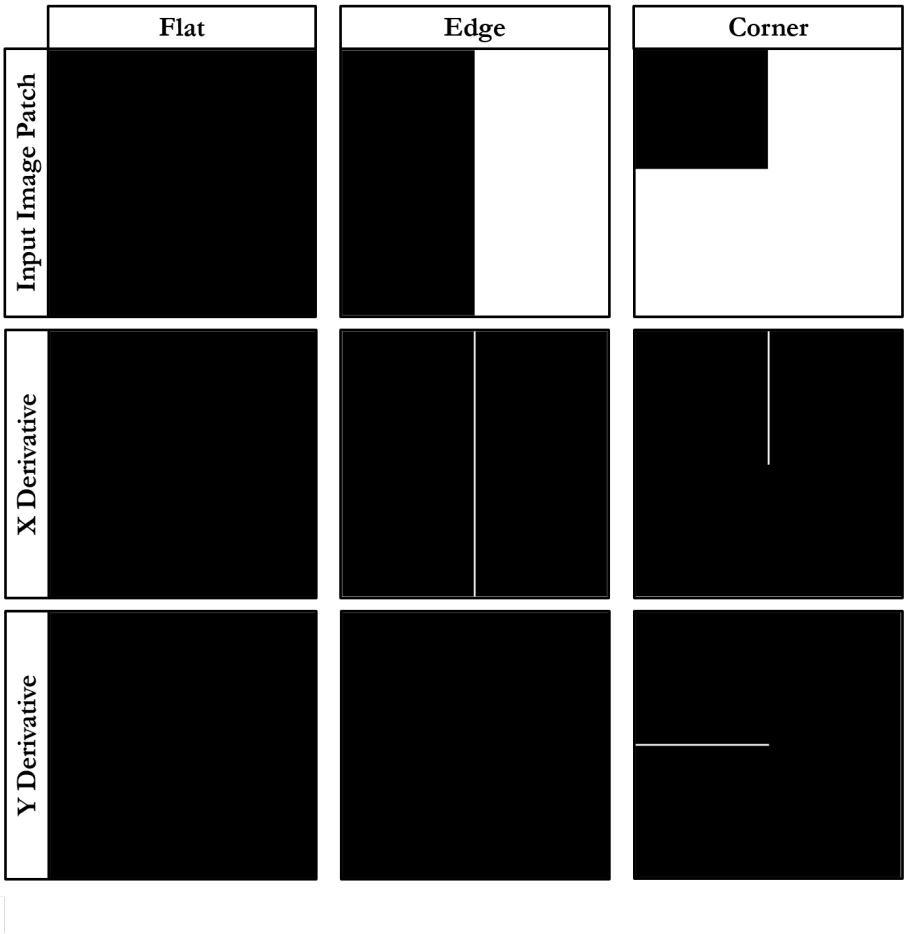


Figure 3.27: Image Gradient Example.

Insert Text Explaining Math.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{3.18}$$

$$\sum_{x,y} [I(x+u, y+v) - I(x,y)]^2 \tag{3.19}$$

$$E(u,v) \approx \sum_{x,y} [I(x,y) + uI_x + vI_y - I(x,y)]^2 \tag{3.20}$$

$$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2 \tag{3.21}$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.22}$$

$$E(u,v) \cong \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.23}$$

Where

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Insert Text Explaining How Both Harris Corner and Good Feature Scoring Functions Work and Why Good Feature Performs Slightly Higher.

$$R = \det(M) - k(\text{trace}(M))^2 \tag{3.24}$$

Where $\det(M) = \lambda_1 \lambda_2$ and $\text{trace}(M) = \lambda_1 + \lambda_2$. So,

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{3.25}$$

$$R = \min(\lambda_1, \lambda_2) \tag{3.26}$$

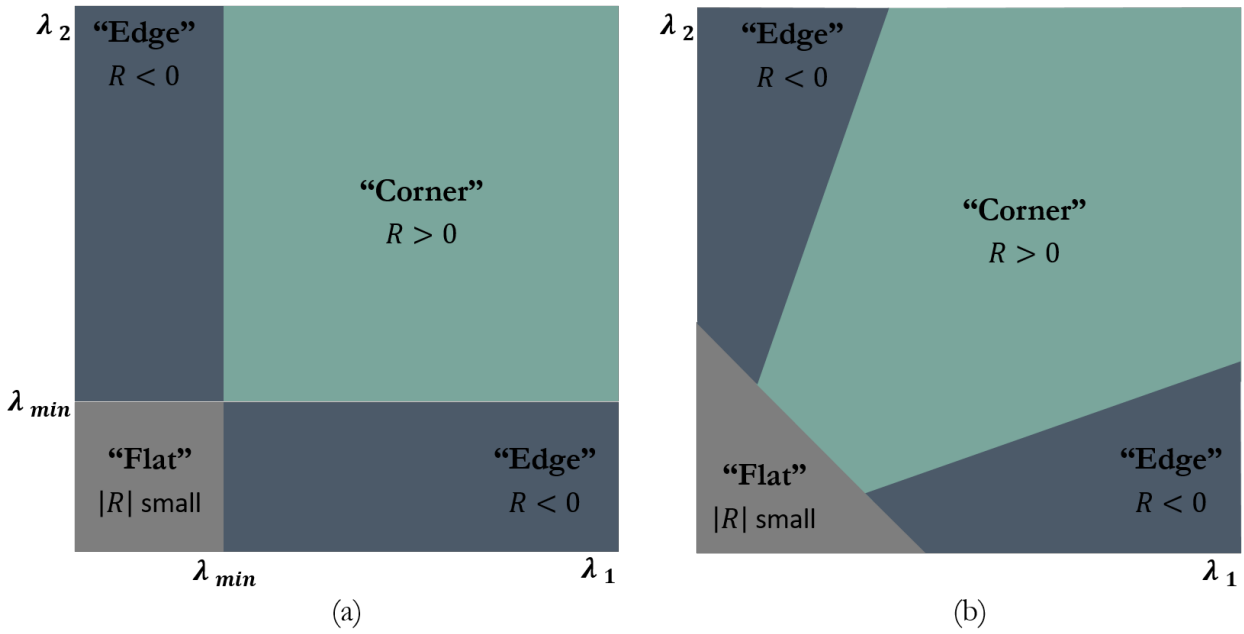


Figure 3.28: Difference Between Good Feature and Harris Corner Scoring Functions.

3.7.7 Optical Flow

Insert Text Explaining the Basic Intuition Behind Optical Flow.

Figure 3.29: Example of optical flow.

Insert Text Explaining the Basic Intuition Behind Kanade-Lucas-Tomasi Feature Tracker.

$$\sum_x [T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2 \tag{3.27}$$

$$\sum_x \left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \tag{3.28}$$

$$\Delta\mathbf{p} = H^{-1}\sum_x \left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})] \tag{3.29}$$

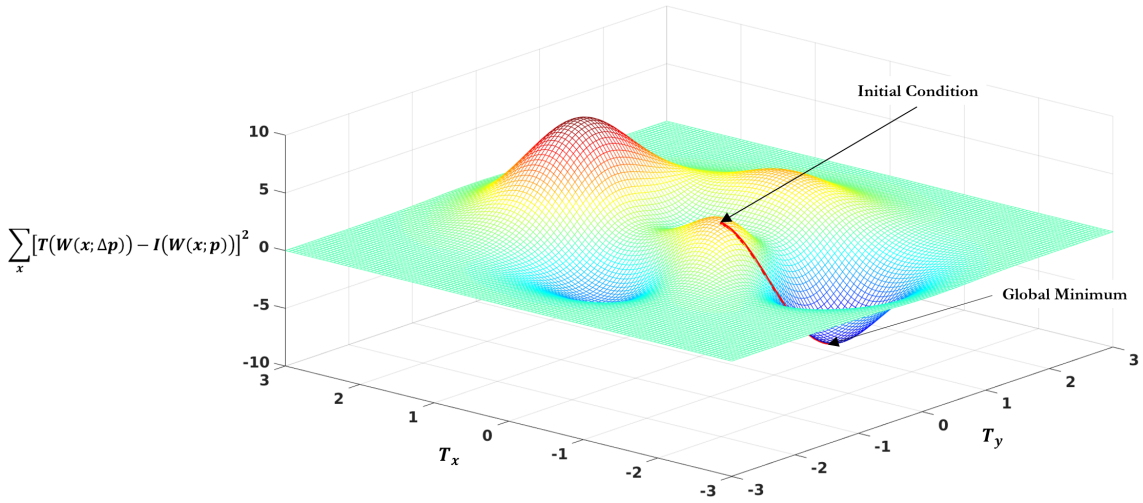


Figure 3.30: Summary of Kanade-Lucas-Tomasi feature tracker.

3.7.8 Pose Estimation and Tracking Through Augmented Reality Tag Detection

AR tags, which are also known as fiducial markers, are artificial landmarks which are designed to be easy to detect and identify under most circumstances [72]. Consequently, the detection method much be robust to changes in luminance (brightness), chrominance (color), and resolution, as well as be scale and rotationally invariant. Once detected and identified, AR tags can be used to accurately calculate the pose (location and orientation) of a camera, as well as to encode information or associate it with each tag's specific identity [73]. Using AR tag to calculate the relative pose of a camera in real-time is called marker-based tracking. Several straightforward and robust maker-based tracking toolkits exists, such as ARToolKit [74], ALVAR [75], and AprilTags [72]. The multistage localization approach presented in this paper will use a ROS implementation of ALVAR [76] to both calculate the pose of an AR tag in the camera's frame and associate information with that tag's specific identity.

Figure 3.31 shows ALVAR's marker-based tracking framework. The system first acquires a greyscale image, either directly or through conversion from another image format. After which, adaptive thresholding, which can handle local changes in luminance, is used to create a binary image. A binary image consists of a background, black pixels, and objects, sections of white pixels. Subsequently, the contours of each object are found. Due to the fact that AR systems, like ALVAR, aim for real-time processing and fast performance, time cannot be wasted processing non-markers. As a result, fast acceptance/rejection tests are run. ALVAR uses two fast accept/reject tests including size and the four-corner test. The number of pixels

in an object's perimeter can be efficiently used to estimate the object's area. Objects with areas that are either too large or too small can be rejected. Even if an object, whose area is too small, is a marker, it is too far from the camera to either be correctly identified or the pose of the camera accurately calculated. Objects, whose area is too large, can be rejected given some background knowledge about the upper and lower range of the camera from the marker, as well as the marker's size. The contours of all remaining markers are fitted with lines. A quadrilateral has exactly four straight lines and four corners. The number of lines and corners of each object are calculated. Those that fail the four-corner test are dropped. After determining that an object is a marker, it is identified and the corner locations are optimized to sub-pixel accuracy for further calculations. Even small errors in the detected 2-D locations of edges and corners can significantly affect the calculated pose of the camera [73]. The pose of a calibrated camera can be uniquely determined from a minimum of four coplanar but non-collinear points [56]. Thus, the system can calculated a marker's pose (relative to camera) in 3-D coordinates using the four corner points of the marker in image coordinates.

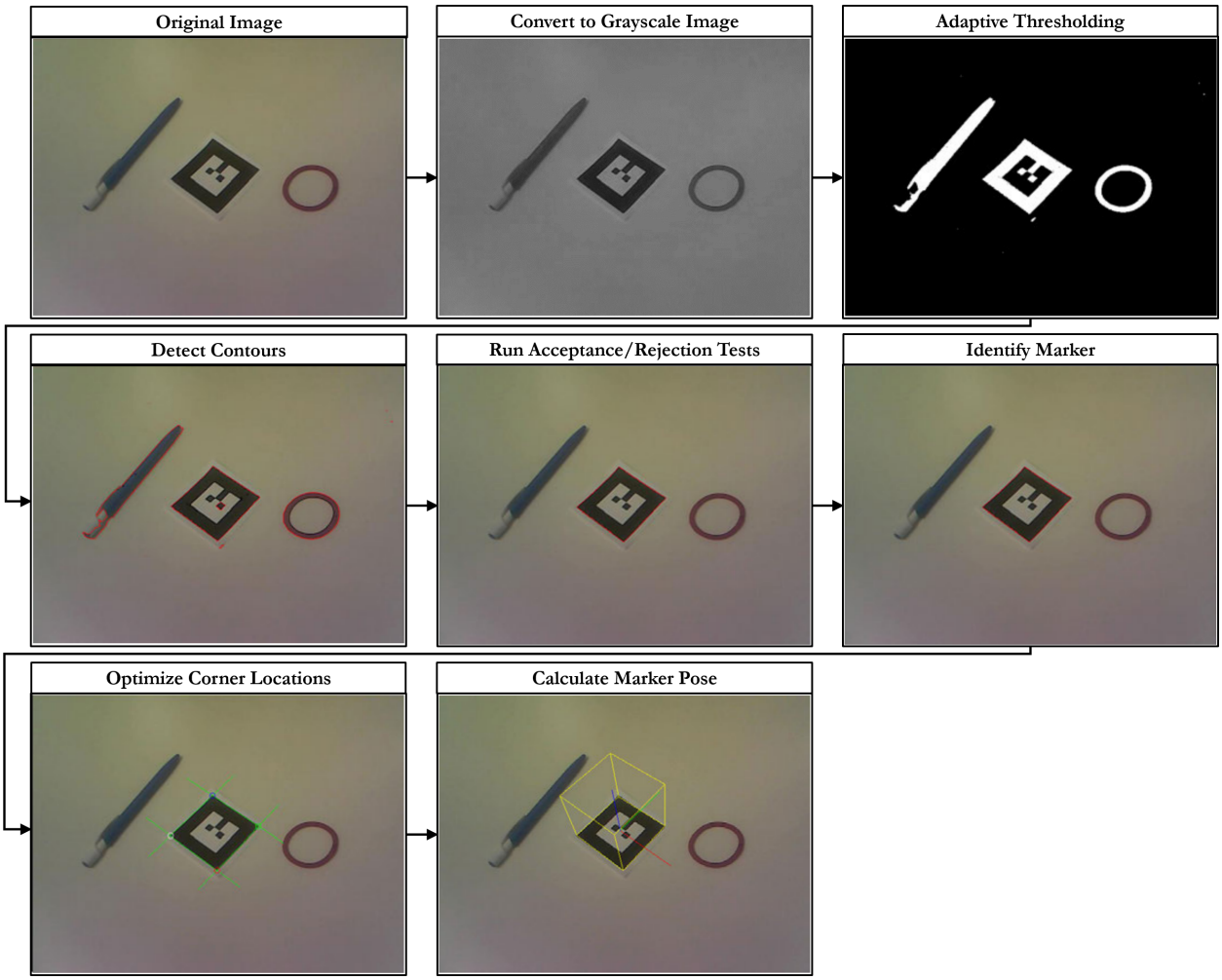


Figure 3.31: Summary of ALVAR AR Tag Detection Framework. Adapted from [73].

3.8 Summary

Insert Text Summarizing This Chapter and Transiting to the Next.

Chapter 4

Multistage Localization for High Precision Mobile Manipulation Tasks

The purpose of this chapter is to thoroughly explain each component of the multistage localization approach presented in this paper, as well as how these individual components fit together in order to enable high precision 3-D feature point localization. In addition, this chapter will expound upon the implementation of this approach to two tasks which are prevalent in both manufacturing and construction, drilling and sealant application operations.

4.1 Approach Overview

Figure 4.1 summaries the multistage localization approach's framework. The mobile manipulator first localizes itself within the area of operation (AO) using adaptive Monte Carlo localization, which relies on the fused odometry and sensor messages published by the robot, as well as a 2-D map of the AO, normally generated using an optimization-based smoothing simultaneous localization and mapping (SLAM) technique. The robot navigates to a predefined start location in the map using a technique called trajectory rollout. Once there, the robot uses an RGB-D sensor to localize an augmented reality (AR) tag in the map frame. One localized, the identity and the 3-D position and orientation, collectively known as pose, of the tag are used to generate a list of initial feature points and their locations based off of a priori knowledge. After the end-effector moves to the rough location of a feature point provided by the AR localization, the feature point's location, as well as the end-effector positon and orientation are refined to within a user specified tolerance through the use of a control loop, which utilized images from a calibrated machine vision camera and a laser pointer, simulating stereo vision, to localize the feature point in 3-D using computer vision.

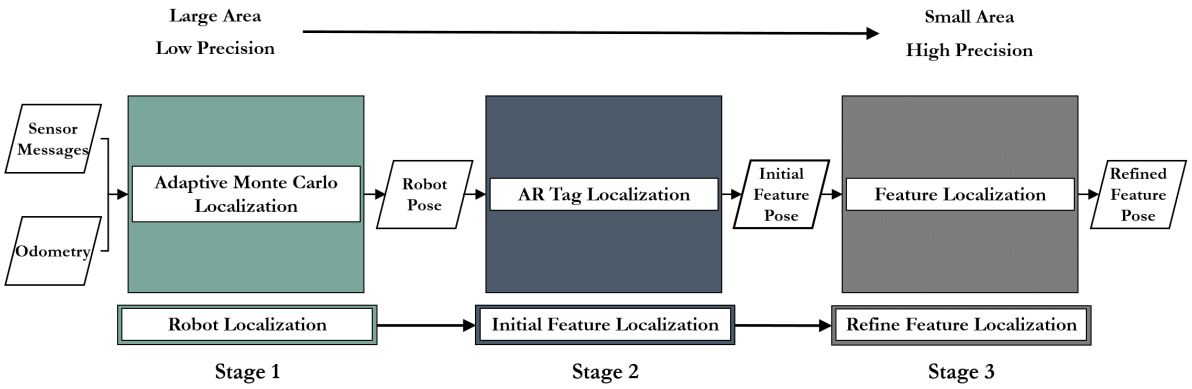


Figure 4.1: Multistage Localization Approach Overview.

4.2 Global Map Creation and Task Location Specification

In order for a mobile manipulator to operate and perform intricate tasks within a complex, GPS-denied environments, such as that of manufacturing facilities or construction site, without modifying said environment, the robot must create an accurate map of its environment. Simultaneous localization and mapping (SLAM) is the process of building or updating a map of an unknown environment while simultaneously localizing the robot within this map. Most SLAM techniques can be categorized into two main paradigms: filtering and optimization-based smoothing. While filtering SLAM techniques have been widely used in the past due to their ability to model different sources of sensor noise, in recent years optimization-based smoothing techniques have proven to be more efficient, scalable, and robust than that of filtering techniques [14]. ROS [1], through which the multistage localization approach was implemented, includes several 2-D SLAM packages. Santos et al. performed a thorough real world evaluation of these available 2-D SLAM techniques in [26], in which KartoSLAM [22] [15], an optimization-based smoothing SLAM technique, outperformed the other ROS implemented SLAM algorithms in terms of accuracy and computational efficiency. As a result, optimization-based smoothing techniques, such as KartoSLAM, should be used to generate a 2-D occupancy grid map of the robot's environment. After which, start positions and orientations of the specific operations are defined in the map as seen in Figure 4.2.

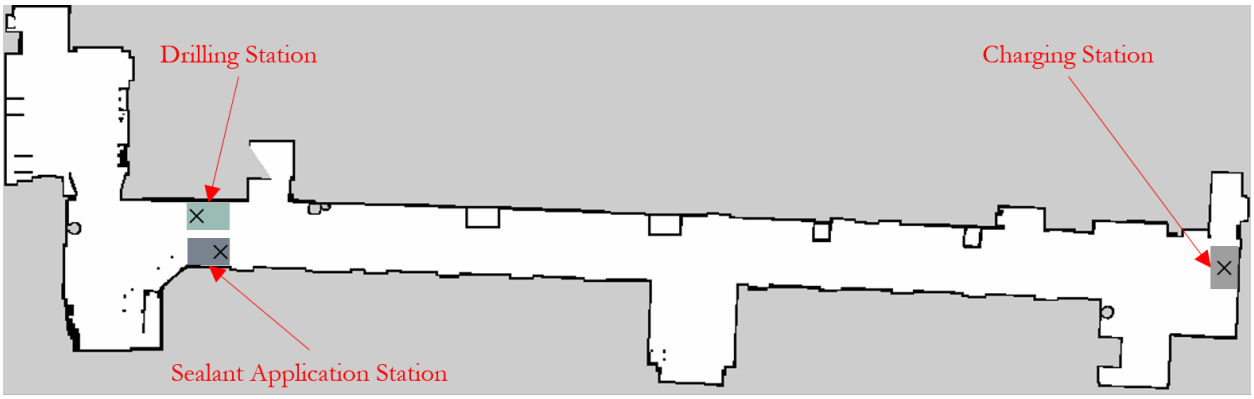


Figure 4.2: 2-D map environment with specified start locations.

4.3 Autonomous Localization and Navigation

After building a 2-D occupancy grid map, shown in Figure 4.2, using an optimization-based smoothing SLAM technique, it becomes crucial to accurately localize the robot within this predefined map; so that, the robot can both plan and execute appropriate trajectories to reach its destination. Figure 4.3 shows how the multistage localization approach localizes the robot within the map frame. Through the measurement of wheel rotation and the integration of accelerations provided by an inertial measurement unit (IMU), the distance traveled by the robot from its initial position can be calculated and the robot's pose in the map estimated. However, these methods do not account for wheel slippage or measurement error. As a result, the accuracy of the pose estimation will degrade over time. Consequently, the odometry measurements from these two sources are fused together through an Extended Kalman filter to provide an accurate pose estimate [77]. The corrected odometry, as well laser scan measurements from a LIDAR and the prebuilt map are used by Adaptive Monte Carlo Localization (AMCL) [29] to localize the robot within this prebuilt map. AMCL utilizes a particle filter to keep track of the robot's pose and *KLD-sampling* to improve computational complexity by removing redundant particles.

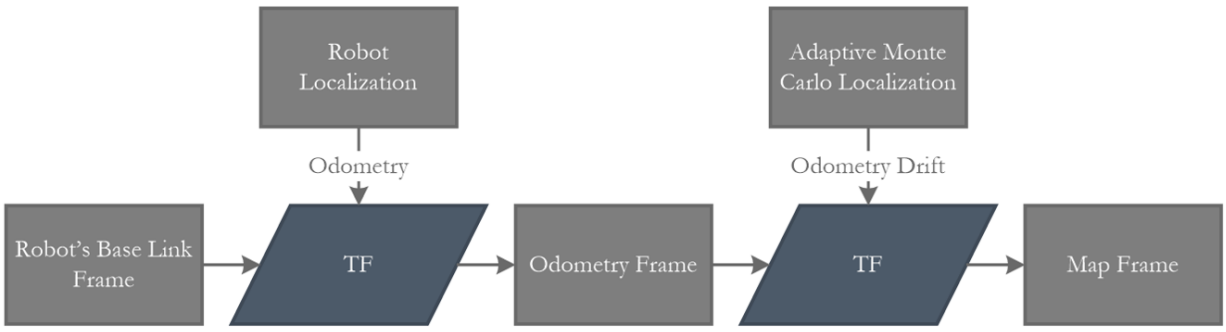


Figure 4.3: Robot Localization Framework.

After localizing the robot within the predefined map, and computing local and global costmaps, in which obstacles and a specified distance around them represent a cost, the robot must now both plan and execute appropriate trajectories to reach its destination. Trajectory Rollout [33] samples the space of feasible controls. For a differential drive robot, this controls space is 2D and consists of translations and rotational velocities, $\dot{x}, \dot{\theta}$, which are limited by the robot's capabilities. This sampled velocity is forward simulated from the robot's current position for a short period of time in order to generate simulated trajectories. These simulated trajectories are then scored using the cost function in Equation 4.1,

$$C(k) = \alpha Obs + \beta Gdist + \gamma Pdist + \delta \frac{1}{\dot{x}^2} \tag{4.1}$$

where *Obs* is the sum of grid cell cost through which the trajectory passes, *Gdist* and *Pdist* are the estimated shortest distance from the endpoint of the trajectory to the goal and the optimal path, respectively, and $\dot{x}$ is the translation component of the velocity command that produces the trajectory. The simulated trajectory that minimizes this cost function is chosen. As a result, chosen trajectories tend to keep obstacles at a distance, proceed towards the goal, remain near the optimal path, as well as have higher velocities [32].

ROS's Navigation Stack [27] is a collection of packages, which the aforementioned techniques can be implemented in, in order to localize the robot within a prebuild map, as well as plan and execute trajectories by outputting the velocity commands needed; so that the robot can reach the specified goal.

4.4 Initial Feature Localization Framework

Each Augmented Reality (AR) tag has a unique identity. As a result, relevant information, such as feature locations relative to the AR tag's pose for a specific operation, can be

associated with a predetermined AR tag identity. So, once the robot has arrived at the operation's specified start positions and orientation, the robot uses the ALVAR package [76] and an RGB-D sensor to detect, identify, and calculated the Augmented Reality (AR) tag pose. Upon identification of the AR tag at the operation start position, the feature locations for that operation are populated and then localized on the work surface given the AR tag's pose in the map frame. Figure 4.4 shows this initial feature localization framework.

Figure 4.4: Initial Feature Localization Framework.

4.5 Motion Planning

Once the operations feature locations have been populated and localized on the work surface in the map frame, the manipulator must be able to move the end-effector to these locations.The manipulator's motion planning is done through ROS's Moveit! package [34], which was set up to use TRACLabs numerical inverse kinematic (IK) solver, TRAC_IK [47]. Due to the recursive nature of numerical solvers, they tend to be slower than that of analytical solvers and are prone to get trapped in local minima. However, analytical solvers require a separate kinematic solver be created for each manipulator and manipulator configuration. Consequently, in order to ensure cross platform functionality with minimum overhead, numerical solvers were used. Orocos Kinematics and Dynamics Library (KDL) [45], Moveit! default numerical solver, as well as arguably the most widely-used generic numerical solver worldwide, suffers from an unacceptably high failure rate and slow average speed. Beeson et al. determined in [48] the causes behind the high failure rates and slow average speed, as well as developed TRAC-IK to overcome them. Beeson et al. revealed KDL had following flaws:

1. Frequent convergence failures due to manipulators with joint limit
2. No action is taken when the search gets trapped in a local minima
3. Lack of tolerance support and/or utilization in the solver itself

TRAC-IK solves these issues by using two separate IK solvers concurrently, as well as incorporating local minimum detection in both. It uses KDL-RR, a rewrite of KDL which incorporation local minimum detection in order to solve problems two and three by "unsticking" the convergence algorithm through the introduction of random seed angles

when a local minimum is detected, as well as a sequential quadratic programming (SQP) IK implementation, which is an iterative algorithm for nonlinear optimization. The SQP implementation, which incorporates the same local minimum detection as KLD-RR, is able to deal with issue one. However, SQP can have a much longer solve rate than KDL or KDL-RR. As a result, TRAC-IK implements both of these IK methods concurrently and waits for either IK to converge. As a result, the solve rate range between 99.1% and 99.9 % and the average convergence times are well below one millisecond.

4.6 Feature Correction Framework

Unfortunately, localizing features in the map frame based on the AR tag localization is not sufficient for tasks that require very high precision due to errors associated in localizing the AR tag in the map frame. Sources of error include, error in the RGB-D camera's extrinsic calibration, error associated in localizing the AR tag itself, error in the location estimate of the camera with respect to the robot's base link, and finally error in the location estimate of robot's base link in the map. Consequently, a secondary localization method is required to refine the feature pose estimates; so that, they can be used for high precision tasks, such as drilling and sealant application operations in the manufacturing and construction industries.

In order to achieve high precision 3-D localization, the feature must be localized with a high degree of accuracy in the end-effector frame of reference. Consequently, a machine vision camera and a laser pointer, simulating stereo vision, were added to the end-effector. Figure 4.5 show the 3-D localization pipeline developed for high precision feature localization. The 3-D feature localization pipeline runs two separate feature detection algorithms concurrently. The first feature detection algorithm is responsible for locating the center pixel of the laser point produced by the laser pointer. It does this by first converting each RGB frame to the HSV color space. The laser pointer is affixed; so that, the center of the laser point is located within the center region of each frame; so the outside edges of the image are masked. After which, the image is thresholded for the specific color range of the laser point. This produces a binary image, in which only pixels within the specific color range are considered foreground. Morphological filters are used to remove noise, as well as to fill in any holes in the laser point created by the laser pointer; so that, its center can be found. The center is found by finding the centroid of the largest blob in the binary image. The second feature detection algorithm is responsible for locating the pixel of the specific operation's feature. It does this by first converting each RGB frame to gray scale. After which, preprocessing operations, such as a Gaussian and Median filter, are used to minimize the noise in the image without degrading the features. Then Canny Edge Detection [78] is used to find appropriate edges in the image, as well as to produce a binary image, which is used to reduce computational complexity of the feature detection algorithm used. At which point, the outside edges of the image are masked, in order to decrease the likelihood that more than one feature is detected. After which the appropriate feature detection algorithm, such as Hough Circle

Detector [79] or Good Feature to Track [80], is run in order to find the center pixel of the feature closest to the laser point produced by the laser pointer. Once these features' pixel locations has been found, they are tracked using the Kanade Lucas Tomasi (KLT) tracker [81], based on the optical flow of the image, in order to reduce complexation complexity that would be associated with running the aforementioned feature detection algorithms on each frame. The location of center of the laser point can be found by converting the pixel to 3-D camera coordinates using the intrinsic of the camera. However, in order to find the depth the line-line intersection, or midpoint of the shortest line segment between the two lines if the lines do not intersect, between the 3-D ray of the laser pointer in the camera frame, which was found during calibration, and the laser points 3-D ray, which is calculated by finding line connecting the camera's center point and the locations of laser point in camera coordinates on the image plane, must be found. The camera calibration procedure and the calculation of the 3-D ray of the laser pointer in the camera frame are detailed in Section 5.3.1. Given the 3-D location of the laser point in the camera frame, as well as the surface normal of the laser pointer, which was calculated during calibration, the plane that the laser point lies on is calculated. Due to the proximity of the feature to the laser point, they are assumed to lie on the same plane. Consequently, the 3-D locations of the feature point in the camera frame is calculated by finding the line-plane intersection between the 3-D ray of the feature, which is calculated by finding line connecting the camera's center point and the locations of feature point in camera coordinates on the image plane, and the plane of the laser point.

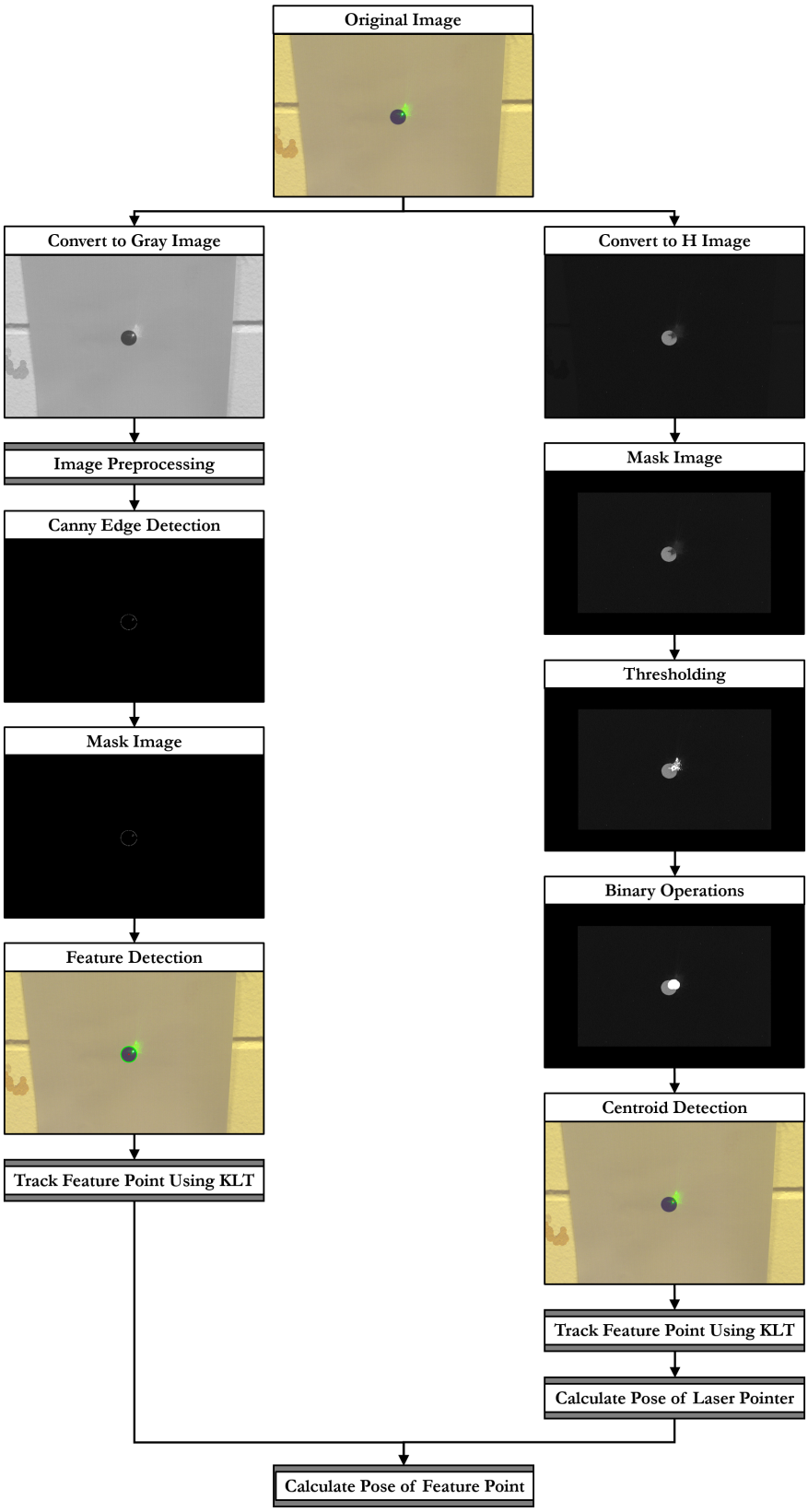


Figure 4.5: Feature Detection and Tracking Pipeline.

After the feature has been localized in 3-D in the camera frame, the feature is transformed into the end-effectors frame and the manipulator updates the position of the end-effector, while remaining a set distance off the work surface, through the use of the control loop depicted in Figure 4.6. Once, the end-effector's is within the reference tolerance set, the control loops exits and the specified operation begins. This reference tolerance should be set no lower than the maximum accuracy achievable by either the detection algorithm or the end-effector itself.

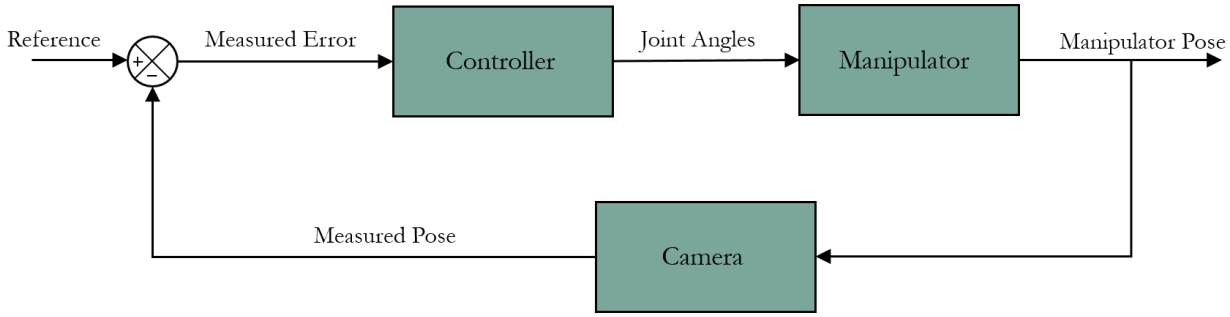


Figure 4.6: Manipulator Correction Control Loop.

4.7 Approach Implementation

The purpose of this section is to briefly explain how the high precision localization approach presented in sections 4.1 through 4.6 can be implemented on a robotic system, as well as to provide an expandable framework; so that, the mobile manipulator can be easily programmed to conduct multiple operations within a manufacturing facility or construction site. Two operations, drilling and sealing, will be explored in depth.

4.7.1 System Overview

The generic framework of the system is depicted in Figure 4.7. The system allows the user to input a number of predefined tasks. Given a priori knowledge of each tasks, as well as their global start locations, the mobile manipulator will navigates to and performs the requested operation(s) while monitoring its battery level to ensure mission completion. The framework shown was implemented using a hierarchical state machine and was written using the Python library SMACH [51] in order to ensure compatibility with ROS; thereby, ensuring seamless integration with the aforementioned packages, as well as ease of use across differing robotic platforms.

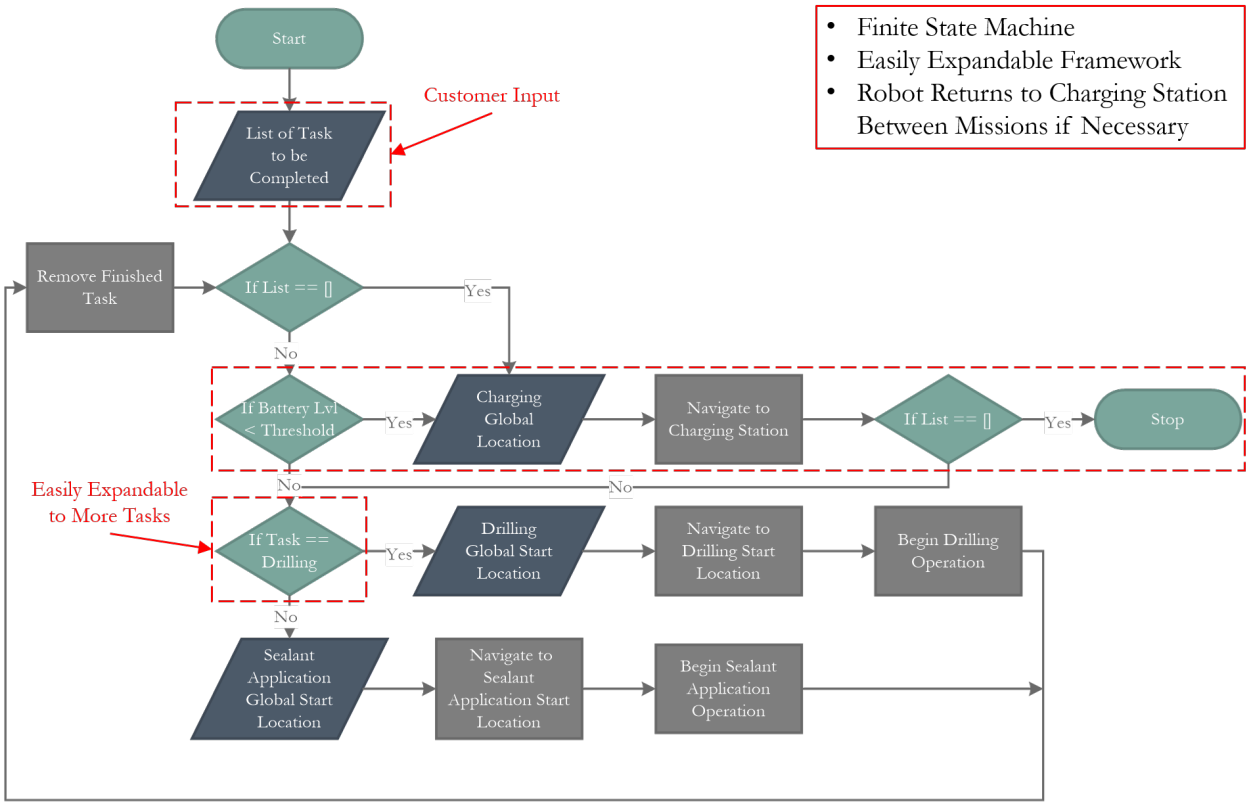


Figure 4.7: General system overview.

4.7.2 Drilling Framework

Figure 4.8 shows the basic drilling operation and error handling framework. Once the mobile manipulator has arrived at the drilling operation's start location and orientated itself in the prescribe direction, its RGB-D camera will detect, identify, and localize the AR tag located at the work station. Using this AR tag locations and identity, the specific features associated with that work station will be populated and localized on the work surface in the map frame. After these initial feature locations have been populated, the robot will move within range of the first feature location in the list. The manipulator will move the end-effector to specified Cartesian location specified using Moveit! and TRAC-IK numerical solver. Currently, many companies use hole templates to ensure that objects are drilled within necessary tolerances. Consequently, the feature correction framework will use these features as landmarks and detect them using Hough Circle Detector [79]. Using these landmarks, the feature locations are refined to within a user specified accuracy. After which, the manipulator will drill the hole to the specified depth and at the specified feed rate requested by the user. This will continue until all features have been drilled. However, if the robot due to obstruction or

other reason is unable to navigate within range of a feature, the machine vision camera is unable to detect the landmark, or the manipulator is unable to reach the location, these specified features will be added to a flag list and the customer notified after the operation.

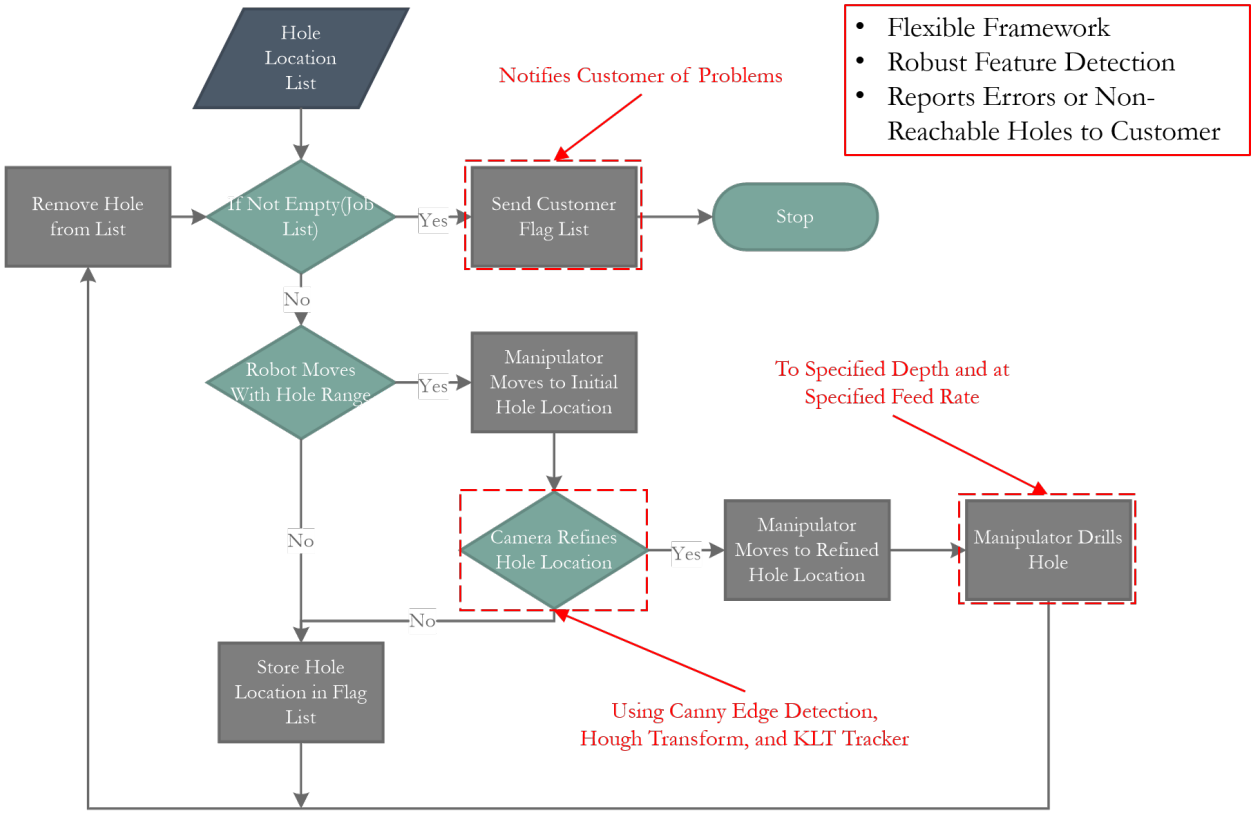


Figure 4.8: Drilling operation framework.

4.7.3 Sealant Application Framework

Figure 4.9 shows the basic sealant application operation and error handling framework. Once the mobile manipulator has arrived at the sealant applicant operation's start location and orientated itself in the prescribe direction, its RGB-D camera will detect, identify, and localize the AR tag located at the work station. Using this AR tag locations and identity, the specific features associated with that work station will be populated and localized on the work surface in the map frame. After these initial feature locations have been populated, the robot will move within range of the first feature location in the list. The manipulator will move the end-effector to specified Cartesian location using Moveit! and TRAC-IK numerical solver. Most sealant applicant operations, such as sealing a window frame, require the application of sealant from one corner to another. Consequently, the feature correction

framework will use these features, corners, as landmarks and detect them using Good Feature to Track [80]. Using these landmarks, the feature locations are refined to within a user specified accuracy. After which, the manipulator will move to the next corner location in the list. After this corner's location has been refined, the manipulator will transition back to the previous corner's location and seal from that start corner to the end corner. This will continue until all connected corners have sealant applied between them. However, if the robot due to obstruction or other reason is unable to navigate within range of a feature, the machine vision camera is unable to detect the landmark, or the manipulator is unable to reach the location, these specified features will be added to a flag list and the customer notified after the operation.

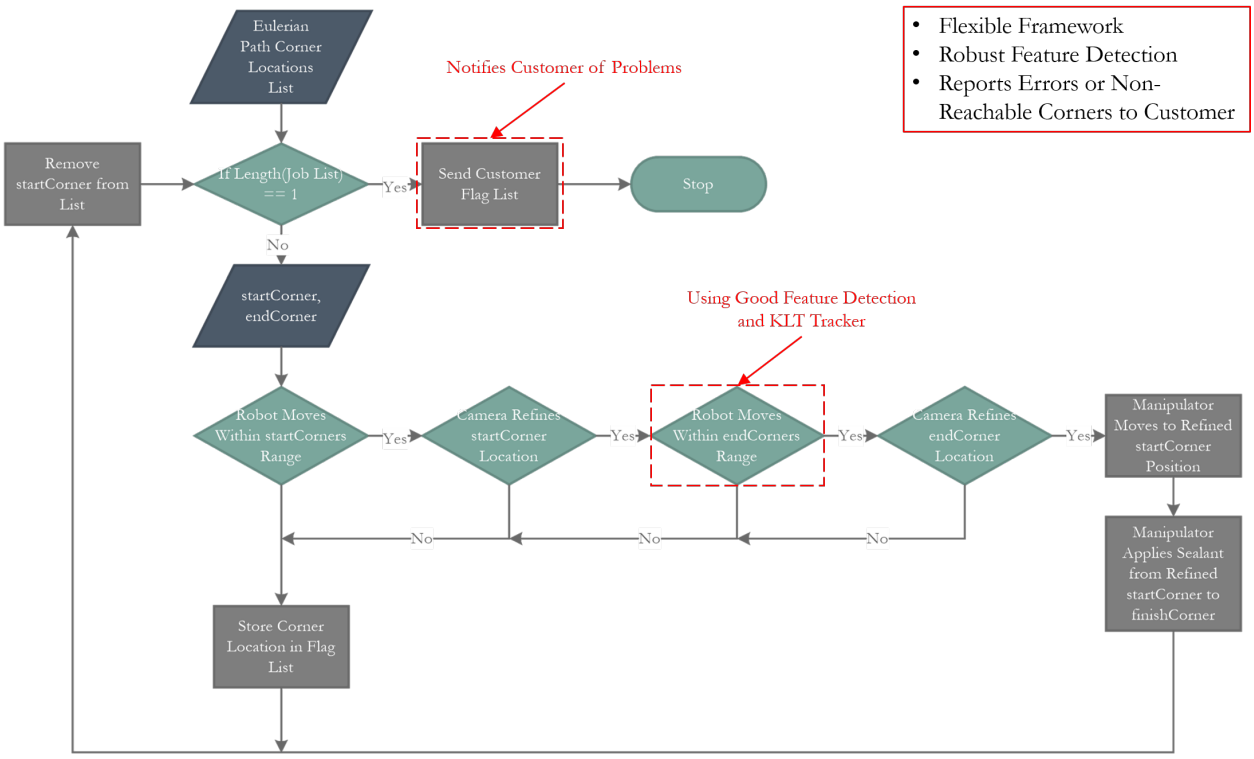


Figure 4.9: Sealant Application Framework.

4.8 Summary

The purpose of this chapter was to thoroughly explain each component of the multistage localization approach presented in this paper, as well as how these individual components fit together in order to enable high precision 3-D feature point localization. In addition, this chapter expounded upon the implementation of this approach to two tasks which

are prevalent in both manufacturing and construction, drilling and sealant application operations. The following chapter will discuss the experiment conducted in order to validate the approach using the two aforementioned operations.

Chapter 5

Experiments and Results

The purpose of this chapter is to validate each stage of multistage localization approach and system implementation presented in Chapter 4 through testing. The accuracy of each component of the multistage localization technique will be determined on two different ROS enabled robotic platforms, the Clearpath Robotics' Husky and Fetch Robotics' Fetch. In addition, these accuracies will be discussed and methods to improve them presented.

5.1 Hardware Architecture

Figure 5.1 shows the Clearpath Robotics' Husky and Fetch Robotics' Fetch with the necessary equipment to simulate high precision operations, such as drilling and sealing. Each robot is equipped with an onboard computer, as well as an RGB-D and RGB camera, a laser pointer, LIDAR, IMU, wheel encoders, and a manipulator.

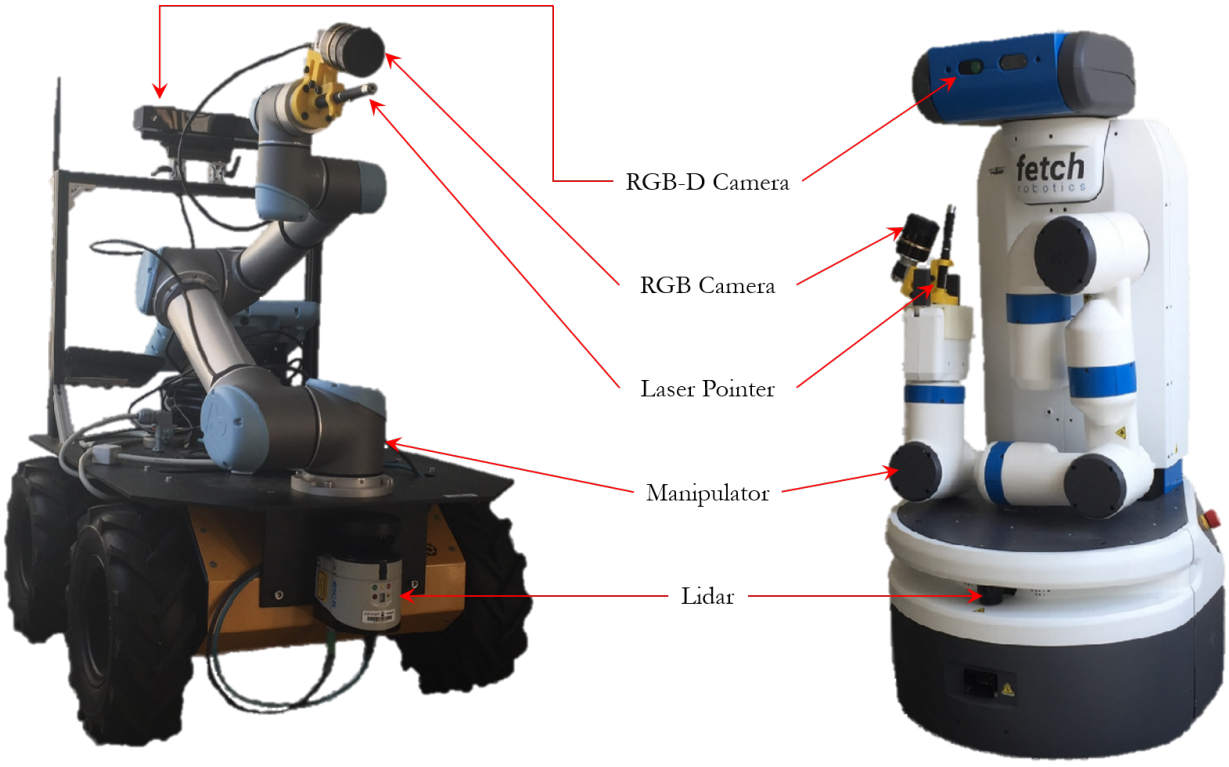


Figure 5.1: Clearpath Robotics' Husky and Fetch Robotics' Fetch Mobile Manipulator

The Clearpath Robotics' Husky in particular is equipped with a Microsoft Kinect V2, Baser ACA1920–150uc machine vision camera, Sick LMS–151 LIDAR, MicroStrain 3DM GX3–25 IMU, four high precision wheel encoders, and the Universal Robotics UR5 6 DOF manipulator with a 5kg payload capacity. In regards to the specific computer hardware, the Clearpath Robotics' Husky is equipped with a Gigibyte BRIX GB-BXi7G3–760 which includes an Intel i7 processor, NVIDIA GeForce GTX 760 GPU, 16 GB DDR3 RAM, 250 GB SSD, and two USB3.0 ports.

The Fetch Robotics' Fetch in particular is equipped with a Primesense Carmine 1.09, Baser ACA1920–150uc machine vision camera, Sick TIM571 LIDAR, two 6 DOF IMUs, two high precision wheel encoders, and Fetch Robot's proprietary 7 DOF manipulator with a 6 kg payload capacity. In regards to the specific computer hardware, The Fetch Robotics' Fetch is equipped with a standard Mini-ITX motherboard, which includes an Intel i5 processer, 16 GB DDR3 RAM, 120 GB SSD, and three USB 2.0 ports.

5.2 Software Architecture

The Clearpath Robotics' Husky and Fetch Robotics' Fetch used Ubuntu 14.04 and ROS Indigo for the experiments. The Clearpath Robotics' Husky used an Extended Kalman Filter through ROS's robot_localization package [82] to fuse its wheel encoder and IMU measurements together, while Fetch Robotics' Fetch used an Unscented Kalman Filter. Both platforms used this corrected odometry, as well as laser scans from their respective LIDAR to produce a map of the area of operation (AO) using ROS's KartoSLAM package [22]. Both robot's use ROS's Navigation Stack [27] to localize themselves in the map using the AMCL package [29] and implement trajectory rollout through the base_local_planner package [33] for path planning and obstacle avoidance. Both robots used ROS's ar_track_alvar package [83] to identify and localize augmented reality (AR) tags. For motion planning, both robot's used ROS's Moveit! [34], which was setup to use the TRAC-IK [47] numerical solver. For perception, both robot's used OpenCV 2.4 [84] to perform necessary operations on the images provided by the Basler machine vision camera.

5.3 Experiments

The purpose of this section is to explain the experimental setup of each test, as well as how the data presented in Section 5.4 was collected. In addition, the camera calibration setup and the software used for the calibration will be presented.

5.3.1 Camera Calibration Setup

Figure 5.2 shows the experimental setup used to calibrate both the Microsoft Kinect V2, the Baser ACA1920–150uc, as well as the laser pointer in the Baser ACA1920–150uc's camera frame. For each calibration the camera and checkerboard pattern were affixed to individual tri-pods. The Microsoft Kinect V2 was calibrated using the kinect2_calibration package [85], while the Baser ACA1920–150 was calibrated using Jean-Yves Bouguet camera calibration toolbox for Matlab [86].

Figure 5.2: Camera Calibration Setup.

The kinect2_calibration package and Bouguet's camera calibration toolbox both follow the calibration procedure proposed by Dr. Zhang in [87]. First, images of a checkerboard pattern affixed to a planar surface are taken by the camera at different distances and orientations. In these images, the feature points in the checkerboard pattern are detected, as depicted in Figure 5.3. These features are used to estimate the five intrinsic parameter of the camera, as well as all the extrinsic parameters using the closed-form equations developed by Dr. Zhang in [87]. The coefficients of radial distortion are then solved using linear least-square. After which, all parameters are refined through nonlinear minimization.

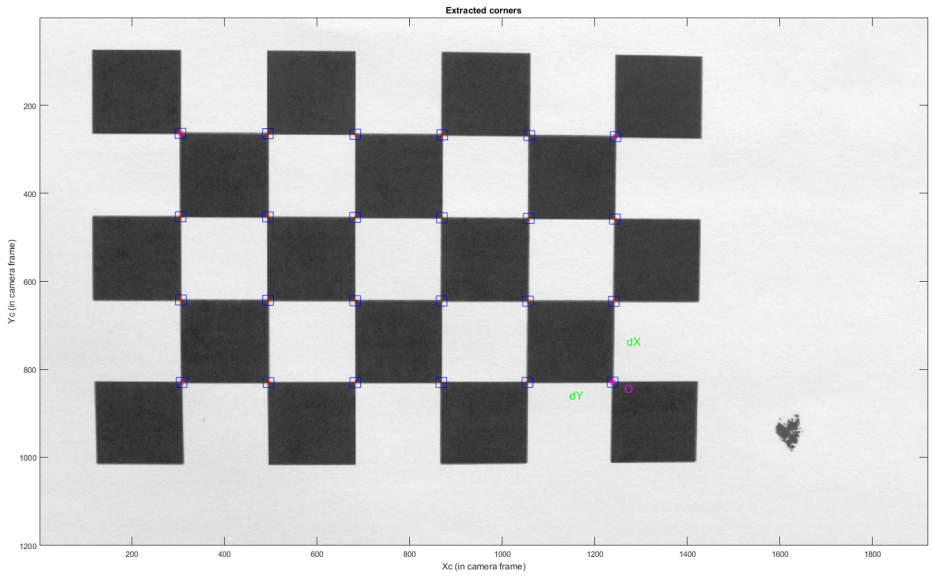


Figure 5.3: Extracted Corners of Calibration Pattern.

After the Baser ACA1920–150 camera had been calibrated, the camera and the laser pointer were affixed to the mount shown in Figure 5.1. After which, the laser pointer 3-D ray equation in camera frame can be calculated by minimizing the error between at least 2 known 3-D coordinates in the camera frame. Thirty points were used to calibrate the 3-D rays equation of the laser pointer for testing. Figure 5.4 shows a checkboard, along with a laser point produced by the laser pointer. The extrinsic parameters of each image are calculated using the feature in the checkerboard. Using the extrinsic, the plane equation of the checkerboard was calculated in camera coordinates. These 3-D camera coordinates points were found by finding line-plane intersection between the plane of the checkerboard and 3-D ray of the laser point, which is calculated by finding line connecting the camera's center point and the locations of laser point in camera coordinates. The same procedure presented in Section 4.6 was used to find the center pixel of the laser point.

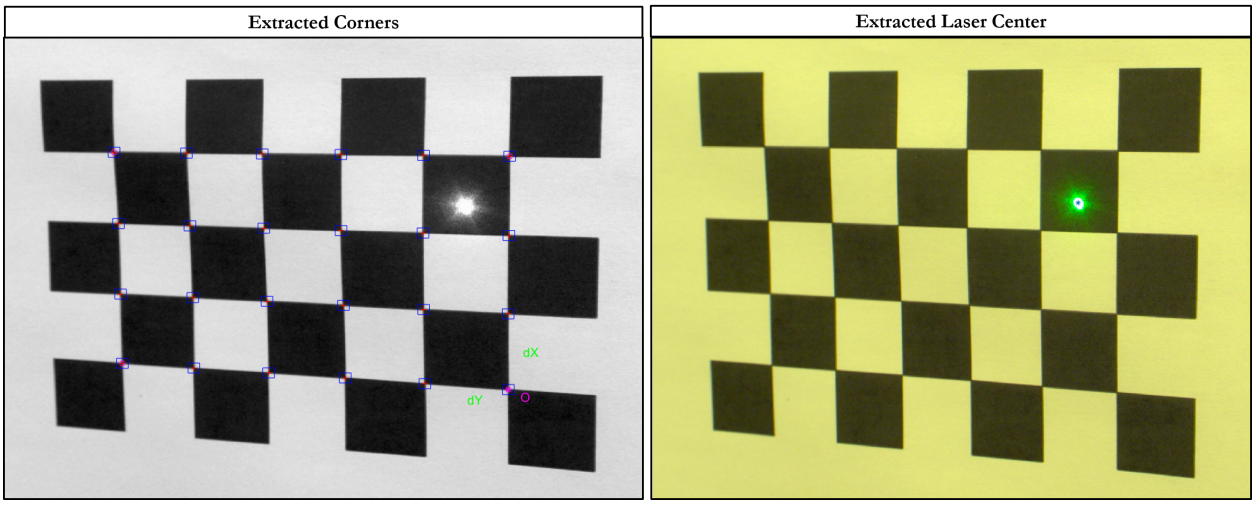


Figure 5.4: Extracted Corners and Laser Center

5.3.2 Navigation System Experimental Setup

Figure 5.5 shows the experimental setup for the navigation system test. Both the Clearpath Robotics' Husky, as well as Fetch Robotics' Fetch navigated to specified positions 5, 10, and 20 meters away from a specified start position. The absolute difference in the x and y directions, as well as the angular yaw between the robot's final positons and the set reference point were measured. This test was run continually as the robot proceed to each 5, 10, and 20 meter location, 20 time each.

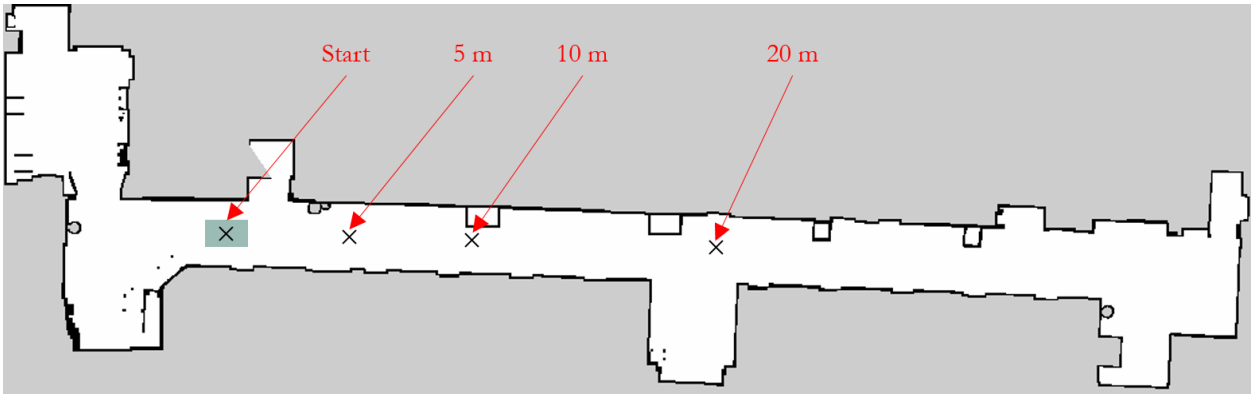


Figure 5.5: Navigation System Experimental Setup.

5.3.3 Augmented Reality Tag Detection, Identification, and Localization System Experimental Setup

Figure 5.6 shows the experimental setup for the augmented reality (AR) tag detection, identification, and localization tests. Both the Clearpath Robotics' Husky, as well as Fetch Robotics' Fetch were placed at locations a specific x and y distance away, as well as at a specific angle from the drilling station's augmented reality (AR) tag, given the maximum error from the navigation test. Each robot's RGB-D sensor was used to detect, identify and localize the specific AR tag. After the feature points for that operation were populated, the absolute difference in x, y, and z directions were measured. For each location, the experiment was conducted five times with 50 features, including the AR tag.

5.3.4 Drill Operation Experimental Setup

Figure 5.6 shows the experimental setup for the drilling operation test. First the sub-millimeter accuracy of the Universal Robotics' 6 DOF UR5 and Fetch Robotics' 7DOF manipulator was verified. After which, the accuracy of feature detection, tracking, and localization pipeline was tested using the Baser ACA1920–150 camera and laser pointer. This accuracy was determined by calculating the difference between the distance measured between the laser point and hole center produced by the feature detection, tracking and localization framework and the ground truth distance between the laser center and hole center measured by hand. This test was performed for 20 locations at each of the following distances between the laser point and hole center: less than 5 mm, between 5 and 10 mm, between 10 and 15 mm, and between 15 and 20 mm.

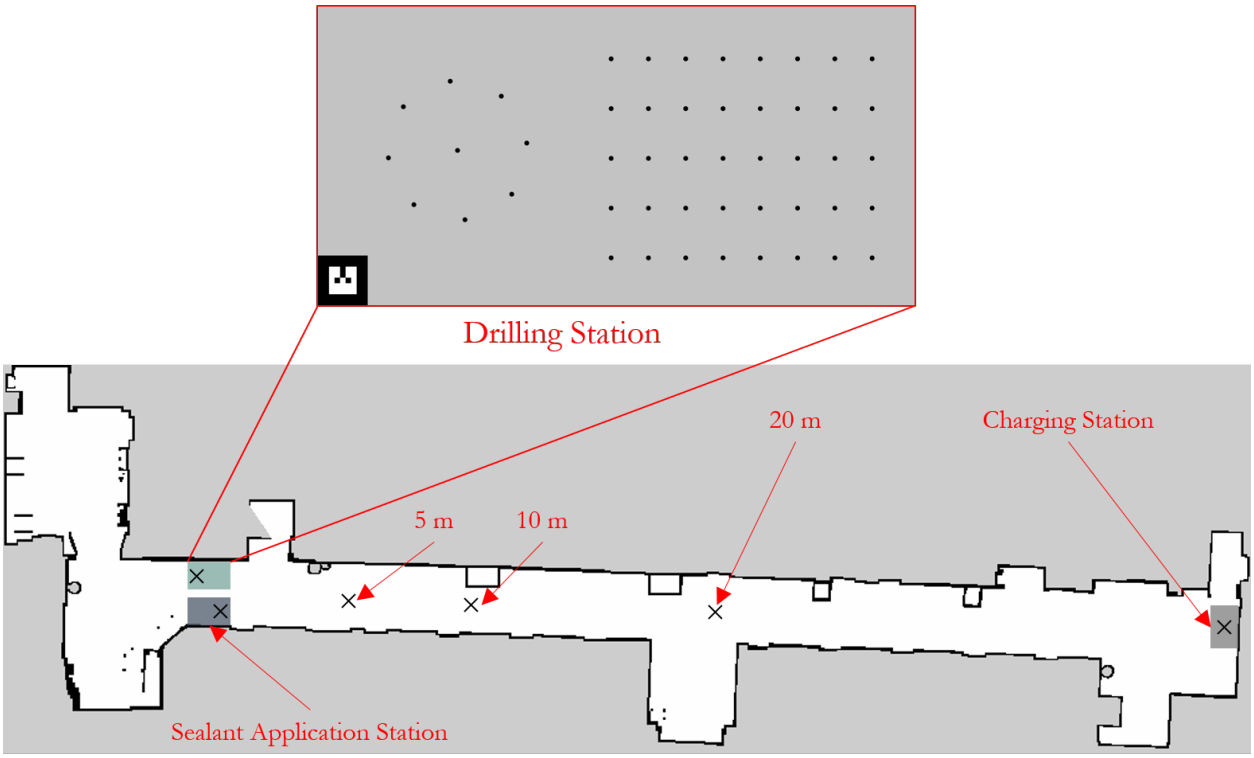


Figure 5.6: Drilling Operations Experimental Setup.

5.3.5 Sealant Application Experimental Setup

Figure 5.7 shows the experimental setup for the sealant operation test. First the sub-millimeter accuracy of the Universal Robotics' 6 DOF UR5 and Fetch Robotics' 7DOF manipulator was verified. After which, the accuracy of feature detection, tracking, and localization pipeline was tested using the Baser ACA1920–150 camera and laser pointer. This accuracy was determined by calculating the difference between the distance measured between the laser point and corner produced by the feature detection, tracking and localization framework and the ground truth distance between the laser center and corner measured by hand. This test was performed for 20 locations at each of the following distances between the laser point and corner: less than 5 mm, between 5 and 10 mm, between 10 and 15 mm, and between 15 and 20 mm.

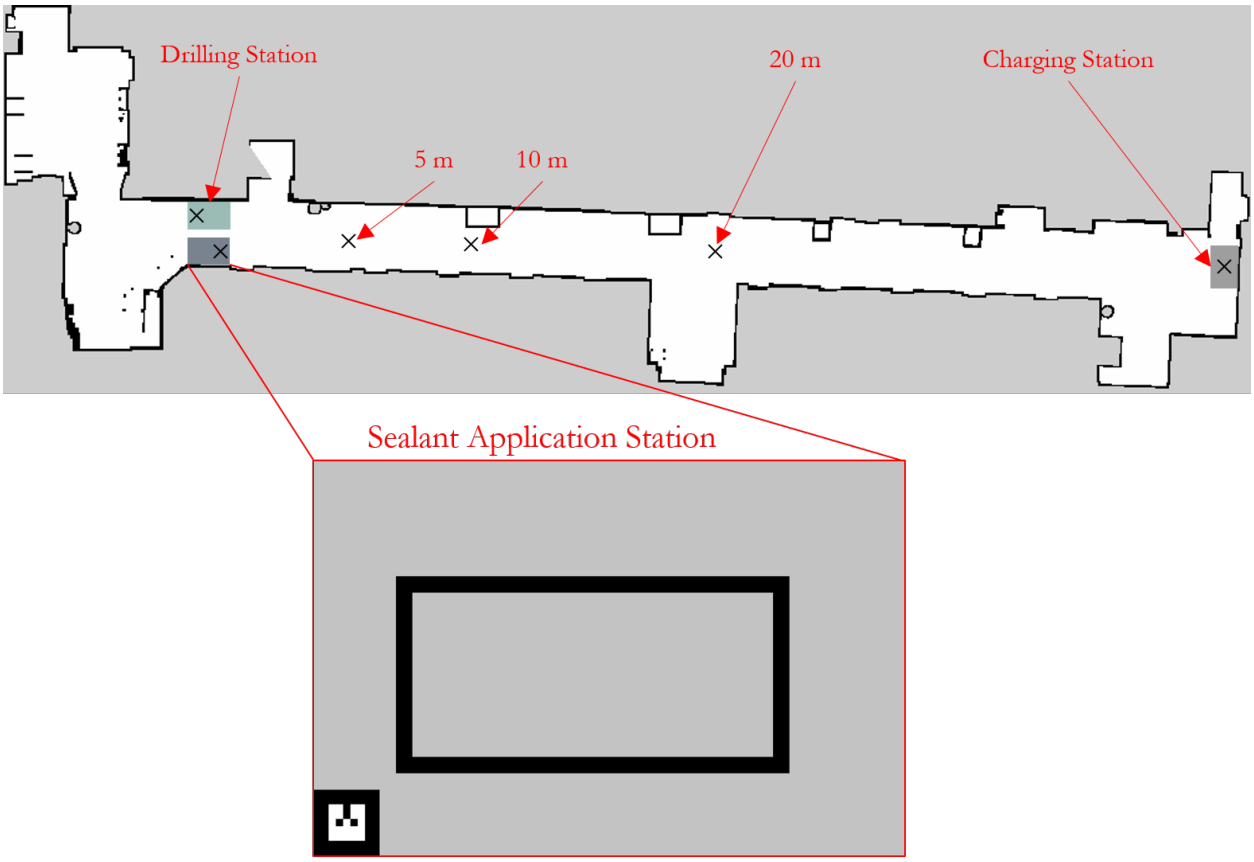


Figure 5.7: Sealant Application Experimental Setup.

5.4 Results

5.4.1 Camera Calibration Accuracy Achieved

Insert Text Regarding Pixel Error of Kinect V2.

Insert Text Regarding Pixel Error of Basler ACA150-UC.

Insert Text Regarding Pixel Error of Laser Calibration.

5.4.2 Navigation System Accuracy Achieved

Insert Text Regarding Accuracy of Navigation System and Ways To Improve Accuracy.

Table 5.1: Global Localization Accuracy Given Distance From Start - Clearpath Robotics' Husky.

	x (cm)	x std (cm)	y (cm)	y std (cm)	ψ (rad)	ψ std (rad)
5 m	6.29	4.20	4.50	3.58	0.03	0.02
10 m	4.16	3.16	2.88	2.97	0.03	0.03
20 m	6.35	4.55	2.37	1.89	0.02	0.01

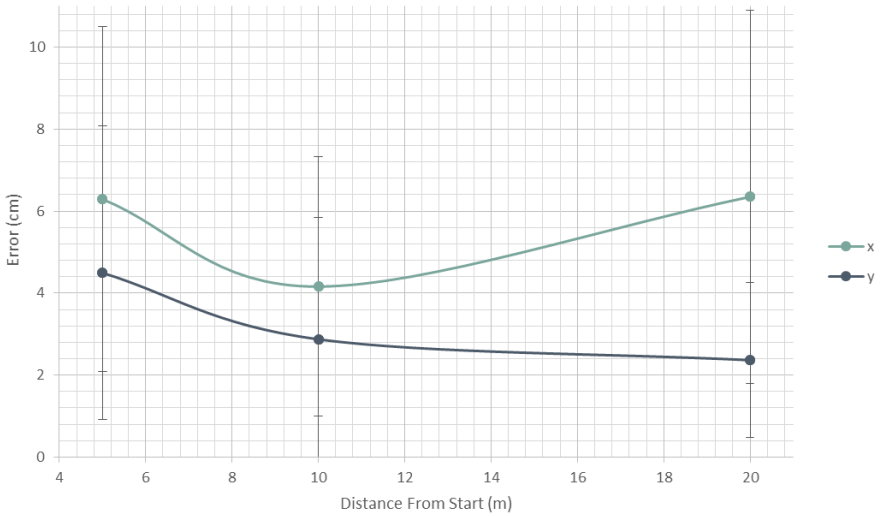


Figure 5.8: Global X and Y Localization Accuracy Given Distance From Start - Clearpath Robotics' Husky.

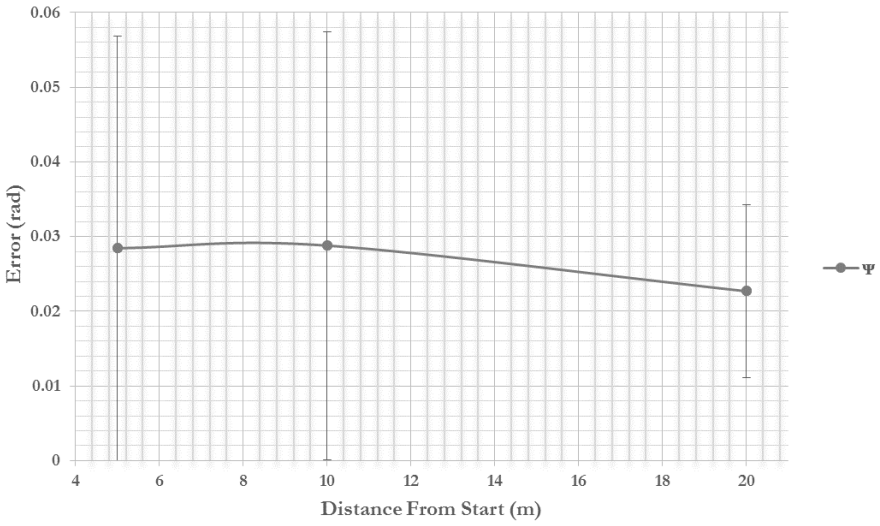


Figure 5.9: Global ψ Localization Accuracy Given Distance From Start - Clearpath Robotics' Husky.

Table 5.2: Global Localization Accuracy Given Distance From Start - Fetch Robotics' Fetch.

	x (cm)	x std (cm)	y (cm)	y std (cm)	ψ (rad)	ψ std (rad)
5 m	4.25	2.50	1.48	1.23	0.03	0.02
10 m	3.93	2.62	2.57	1.66	0.03	0.02
20 m	5.82	3.43	1.52	1.11	0.03	0.02

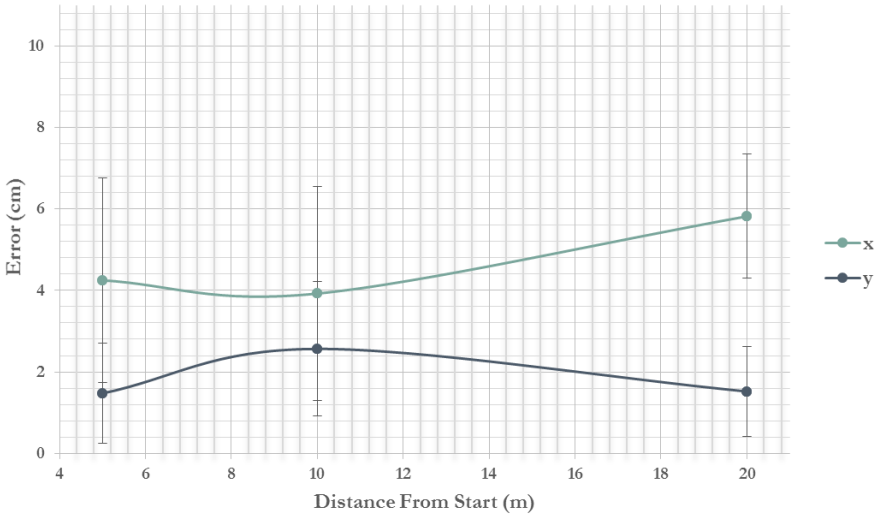


Figure 5.10: Global X and Y Localization Accuracy Given Distance From Start - Fetch Robotics' Fetch.

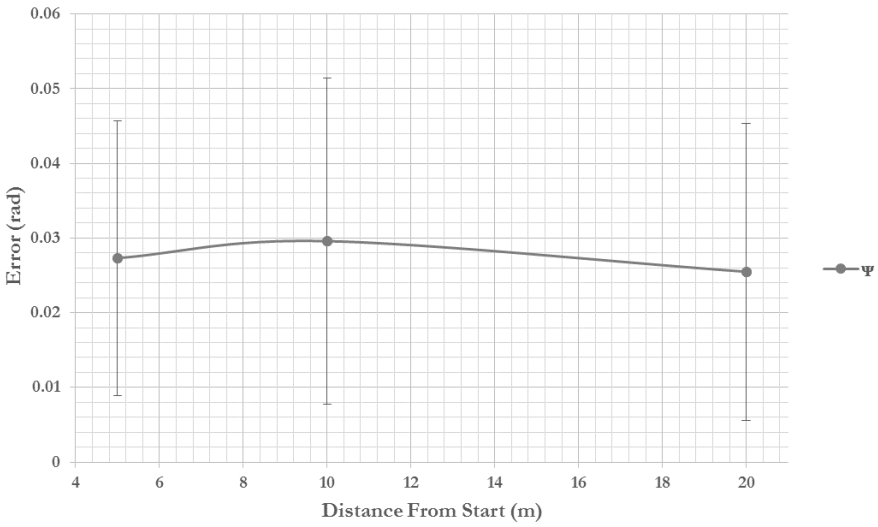


Figure 5.11: Global ψ Localization Accuracy Given Distance From Start - Fetch Robotics' Fetch.

5.4.3 Augmented Reality Tag Accuracy Achieved

Table 5.3: Augmented Reality Accuracy Given Specific Start Conditions - Primesense Carmine 1.09.

	x	x std (mm)	y	y std (mm)	z	z std (mm)
1	50.41	22.81	148.87	12.33	150.96	12.61
2	23.71	16.06	151.82	6.86	117.57	12.81
3	23.06	14.85	128.39	11.48	173.24	23.24
4	37.19	14.85	183.21	7.30	115.85	16.27
5	24.36	17.47	138.66	6.77	42.80	16.37
6	39.16	23.52	7.27	3.70	105.24	9.16

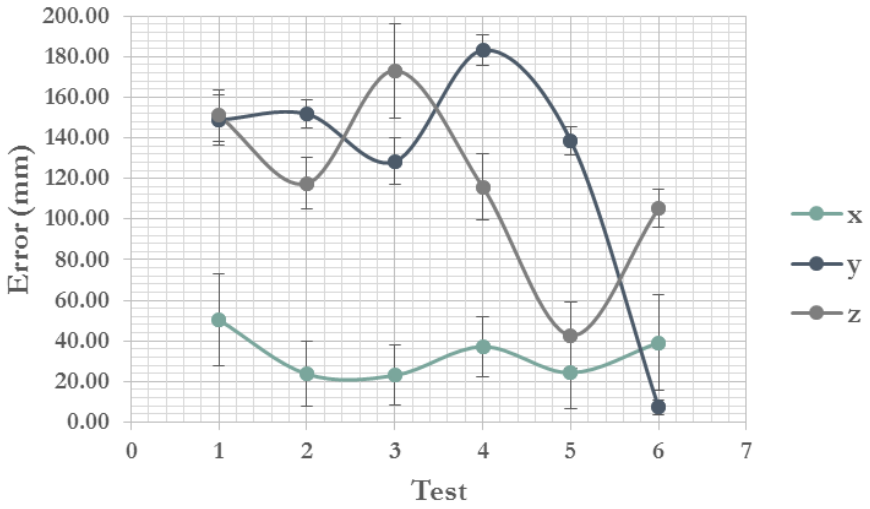


Figure 5.12: Augmented Reality Accuracy Given Specific Start Conditions - Primesense Carmine 1.09.

Table 5.4: Augmented Reality Accuracy Given Specific Start Conditions - Microsoft Kinect V2.

	x (mm)	x std (mm)	y (mm)	y std (mm)	z (mm)	z std (mm)
Test 1	14.32	9.75	4.04	1.65	84.77	2.50
Test 2	163.85	57.58	36.32	9.65	25.31	12.46
Test 3	84.33	44.37	50.31	5.48	55.24	8.57

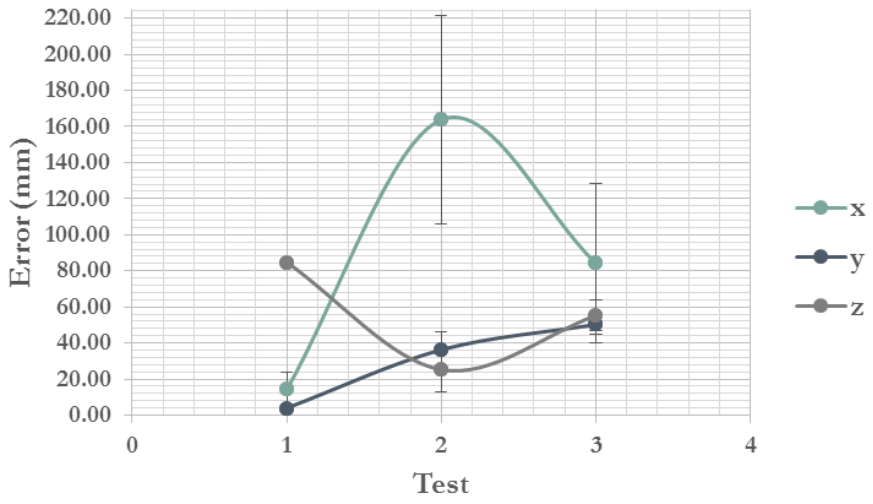


Figure 5.13: Augmented Reality Accuracy Given Specific Start Conditions - Microsoft Kinect V2.

5.4.4 Drilling Operation Accuracy Achieved

Insert Text Regarding Accuracy of Drilling Operations and Ways To Improve Accuracy.

Table 5.5: Drilling Operation Accuracy Given Distance From Work Surface.

	x (mm)	x std (mm)	y (mm)	y std (mm)	z (mm)	z std (mm)
Distance Less Than 5 mm from Hole Center	1.44	1.25	0.70	0.43	0.82	0.56
Distance Between 5 and 10 mm from Hole Center	1.89	1.56	0.66	0.39	1.89	0.46
Distance Between 10 and 20 mm from Hole Center	2.15	1.83	0.53	0.30	4.06	0.70
Distance Between 20 and 30 mm from Hole Center	2.71	2.71	0.31	0.21	10.25	1.38
Mean	2.05	1.53	0.55	0.38	4.25	3.75

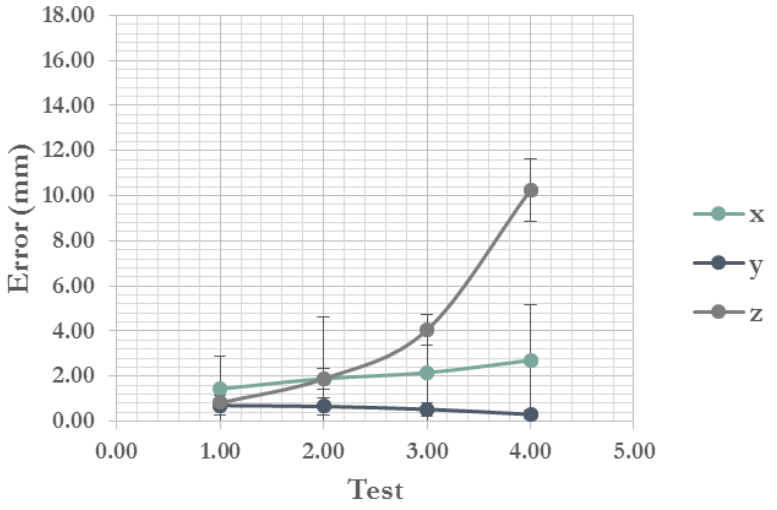


Figure 5.14: Drilling Operation Accuracy Given Distance From Feature Point.

5.4.5 Sealant Application Accuracy Achieved

Insert Text Regarding Accuracy of Sealant Application and Ways To Improve Accuracy.

Table 5.6: Sealant Application Accuracy Given Distance From Work Surface.

	x (mm)	x std (mm)	y (mm)	y std (mm)	z (mm)	z std (mm)
Distance Less Than 5 mm from Corner	2.34	1.29	1.46	0.21	1.17	0.71
Distance Between 5 and 10 mm from Corner	1.60	2.22	1.27	0.22	1.48	0.91
Distance Between 10 and 20 mm from Corner	4.38	2.46	0.78	0.34	3.81	0.32
Distance Between 20 and 30 mm from Corner	3.32	1.95	0.79	0.60	8.21	0.40
Mean	2.91	2.05	1.08	0.48	3.67	2.88

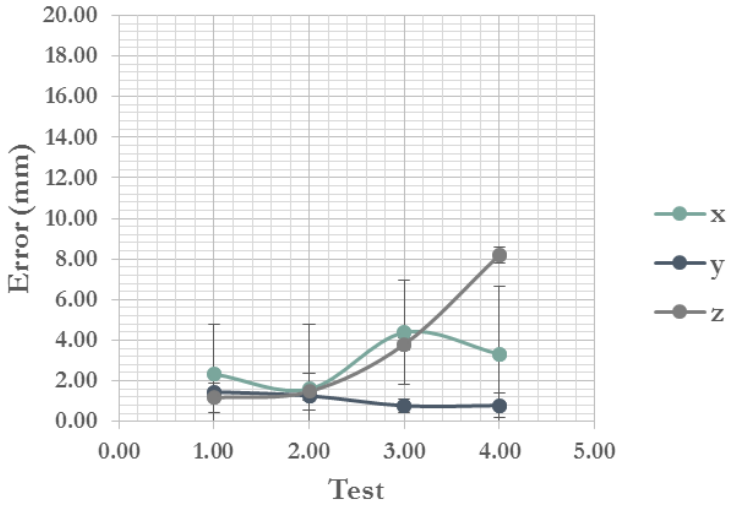


Figure 5.15: Sealant Application Accuracy Given Distance From Feature Point.

5.5 Summary

Insert Text Summarizing This Chapter and Transiting to the Next.

Chapter 6

Conclusion and Future Work

This paper showed the general framework necessary in order to solve the two main problems preventing the development and widespread adoption of ARC. These problems have caused a decrease in productivity and increase in workplace injuries/fatalities over the past several decades in comparison to other industries [**Rojas2003**]. These problems include the fact that typical construction sites tend to be unstructured and are continuously evolving versus the highly controlled environments found in manufacturing. Also, the relationship between the part and manipulator has been reversed, causing increased complexity not seen in manufacturing environments where the part arrives at a fixed manipulator [**Feng2015**]. The techniques presented allow systems to create a 2-D map of their environment, localize themselves and complete the task(s) assigned. After localizing an AR tag at the work site, the system is able to use a priori knowledge to localize POIs and complete a plethora of operations achieving an accuracy of approximately $\pm$ 2 mm based on a multifaceted computer vision approach with only a USB webcam.

Future work to be explored includes increasing the accuracy of the computer vision system to the sub-millimeter levels through the use of a machine vision camera, as well as a relatively new calibration technique developed by Feng et al [**Feng2015**]. In addition, automated approaches to create 3-D maps are being looked into in order to provide updated data about the robot's surroundings and task(s) automatically without human intervention. Also, human and robot collaboration over a distributed network is being explored.

Bibliography

[1] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[2] *ROS.org - Powering the world's robots*. URL: `http://www.ros.org/` (visited on 10/18/2016).

[3] *ROS Concepts - ROS Wiki*. URL: `http://wiki.ros.org/ROS/Concepts` (visited on 10/18/2016).

[4] James David Burton. "Development and Characterization of an Interprocess Communications Interface and Controller for Bipedal Robots". PhD thesis. 2015.

[5] Alonzo Kelly. *Mobile Robotics: Mathematics, Models, and Methods*. New York, NY, USA: Cambridge University Press, 2013.

[6] *TF - ROS Wiki*. URL: `http://wiki.ros.org/tf` (visited on 10/19/2016).

[7] *TF2 - ROS Wiki*. URL: `http://wiki.ros.org/tf2` (visited on 10/19/2016).

[8] *URDF - ROS Wiki*. URL: `http://wiki.ros.org/urdf` (visited on 10/19/2016).

[9] *OpenSLAM - What is SLAM?* URL: `http://openslam.org/` (visited on 10/28/2016).

[10] J J Leonard and H F Durrant-Whyte. "Simultaneous map building and localization for an autonomous mobile robot". In: *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*. 91. 1991, 1442–1447 vol.3.

[11] Randall C Smith and Peter Cheeseman. "On the Representation and Estimation of Spatial Uncertainty". In: *The International Journal of Robotics Research* 5.4 (1986), pp. 56–68.

[12] R. Smith, M. Self, and P. Cheeseman. "Estimating Uncertain Spatial Relationships in Robotics". In: *Autonomous Robot Vehicles* 4.April (1990), pp. 167–193.

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

[14] Doaa M. a Latif et al. "Comparison of Optimization Techniques for 3D Graph-based". In: *Proceedings of the 4th European Conference of Computer Science (ECCS '13) Recent Advances in Information Science*. October. 2013, p. 288.

[15] Kurt Konolige et al. "Efficient sparse pose adjustment for 2D mapping". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*. 2010, pp. 22–29.

[16] *HectorSLAM - ROS Wiki*. URL: `http://wiki.ros.org/hector%7B%5C_%7Dslam` (visited on 10/28/2016).

[17] Stefan Kohlbrecher et al. "A flexible and scalable SLAM system with full 3D motion estimation". In: *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*. 2011, pp. 155–160.

[18] *CoreSLAM - ROS Wiki*. URL: `http://wiki.ros.org/coreslam` (visited on 10/28/2016).

[19] *GMapping - ROS Wiki*. URL: `http://wiki.ros.org/gmapping?distro=kinetic` (visited on 10/28/2016).

[20] Bruno Steux and Oussama El Hamzaoui. "tinySLAM: A SLAM algorithm in less than 200 lines C-language program". In: *11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010*. December. 2010, pp. 1975–1979.

[21] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved techniques for grid mapping with Rao-Blackwellized particle filters". In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46.

[22] *KartoSLAM - ROS Wiki*. URL: `http://wiki.ros.org/slam%7B%5C_%7Dkarto?distro=groovy` (visited on 10/28/2016).

[23] *LagoSLAM - ROS Wiki*. URL: `https://github.com/rrg-polito/rrg-polito-ros-pkg` (visited on 10/28/2016).

[24] Regis Vincent, Benson Limketkai, and Michael Eriksen. "Comparison of indoor robot localization techniques in the absence of GPS". In: *SPIE 7664, Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV*. Ed. by Russell S. Harmon, John H. Holloway, Jr., and J. Thomas Broach. International Society for Optics and Photonics, Apr. 2010, 76641Z.

[25] Luca Carlone and Rosario Aragues. "A linear approximation for graph-based simultaneous localization and mapping". In: *Int. Conf. Robotics: Science and Systems*. 2011, p. 8.

[26] João Machado Santos, David Portugal, and Rui P Rocha. "An Evaluation of 2D SLAM Techniques Available in Robot Operating System". In: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2013.

[27] *Navigation - ROS Wiki*. URL: `http://wiki.ros.org/navigation` (visited on 10/18/2016).

[28] *Map Server - ROS Wiki*. URL: `http://wiki.ros.org/map%7B%5C_%7Dserver` (visited on 11/04/2016).

[29] *AMCL - ROS Wiki*. URL: `http://wiki.ros.org/amcl` (visited on 11/04/2016).

[30] *Move Base - ROS Wiki*. URL: http://wiki.ros.org/move%7B%5C_%7Dbase (visited on 11/04/2016).

[31] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[32] Brian P Gerkey and Kurt Konolige. "Planning and Control in Unstructured Terrain". In: *ICRA Workshop on Path Planning on Costmaps*. 2008.

[33] *Base Local Planner - ROS Wiki*. URL: http://wiki.ros.org/base%7B%5C_%7Dlocal%7B%5C_%7Dplanner (visited on 10/17/2016).

[34] *Moveit Home*. URL: http://moveit.ros.org/ (visited on 10/30/2016).

[35] *RVIZ - ROS Wiki*. URL: http://wiki.ros.org/rviz (visited on 10/30/2016).

[36] *Parameter Server - ROS Wiki*. URL: http://wiki.ros.org/Parameter%20Server (visited on 10/30/2016).

[37] *SRDF - ROS Wiki*. URL: http://wiki.ros.org/srdf (visited on 10/30/2016).

[38] *MoveIt Concepts - MoveIt Documentation*. URL: http://moveit.ros.org/documentation/concepts/ (visited on 10/30/2016).

[39] *Forward Kinematics - Wikipedia*. URL: https://en.wikipedia.org/wiki/Forward%7B%5C_%7Dkinematics (visited on 10/31/2016).

[40] *Inverse Kinematics - Wikipedia*. URL: https://en.wikipedia.org/wiki/Inverse%7B%5C_%7Dkinematics (visited on 10/31/2016).

[41] Patrick Goebel. *ROS By Example Volume 2: Packages and Programs for Advanced Robot Behaviors*. Lulu Enterprises, Inc., 2015.

[42] Behnam Asadi. "Single and Dual Arm Manipulator Motion Planning Library". In: *Workshop on Task Planning for Intelligent Robots in Service and Manufacturing. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Task Planning for Intelligent Robots in Service and Manufacturing, located at IROS 2015*. 2015.

[43] Rosen Diankov and James Kuffner. *OpenRAVE : A Planning Architecture for Autonomous Robotics*. Tech. rep. July. Pittsburgh, PA: Robotics Institute, Carnegie Mellon University, 2008. DOI: CMU-RI-TR-08-34.

[44] *IKFast Module - OpenRAVE Documentation*. URL: http://openrave.org/docs/latest%7B%5C_%7Dstable/openravepy/ikfast/%7B%5C#%7Dikfast-the-robot-kinematics-compiler, (visited on 10/31/2016).

[45] *Orocos KDL - ROS Wiki*. URL: http://wiki.ros.org/orocos%7B%5C_%7Dkdl (visited on 10/31/2016).

[46] *TRACLab - An Exploratory Research Lab Driven by a Curious, Pioneering Spirit*. URL: https://traclabs.com/ (visited on 10/31/2016).

[47] *Trac IK - ROS Wiki*. URL: http://wiki.ros.org/trac%7B%5C_%7Dik (visited on 10/31/2016).

[48] Patrick Beeson and Barrett Ames. "TRAC-IK: An open-source library for improved solving of generic inverse kinematics". In: *IEEE-RAS International Conference on Humanoid Robots*. Vol. 2015-Decem. 2015, pp. 928–935. ISBN: 9781479968855. DOI: 10.1109/HUMANOIDS.2015.7363472.

[49] Hassan Gomaa. *Real-Time Software Design for Embedded Systems*. Cambridge University Press, 2016. ISBN: 9781107041097.

[50] Ian Millington and John Funge. *Artificial Intelligence for Games*. CRC Press, 2016. ISBN: 9781498785815.

[51] *SMACH - ROS Wiki*. URL: http://wiki.ros.org/smach (visited on 10/31/2016).

[52] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 1848829345, 9781848829343.

[53] Ioannis Pitas. *Digital Image Processing Algorithms and Applications*. John Wiley & Sons, 2000. ISBN: 9780471377399.

[54] *How Does a Digital Camera Work?* URL: https://www.google.com/url?sa=i%7B%5C&%7Drct=j%7B%5C&%7Dq=%7B%5C&%7Desrc=s%7B%5C&%7Dsource=images%7B%5C&%7Dcd=%7B%5C&%7Dved=0ahUKEwjy3uuhwovQAhUE24MKHe6DCMIQjxwIAw%7B%5C&%7Durl=http%7B%5C%%7D3A%7B%5C%%7D2F%7B%5C%%7D2Fwww.notey.com%7B%5C%%7D2Fblogs%7B%5C%%7D2Fadc-travels%7B%5C%%7D3Fpage%7B%5C%%7D3D2%7B%5C&%7Dpsig=AFQjCNFBhPHyArfz8DKZ4GolmgBgRXlyTQ%7B%5C&%7Dust=1478225767235938 (visited on 11/02/2016).

[55] Karim Nice, Tracy Wilson, and Gerald Gurevich. *How Digital Cameras Work*. URL: http://electronics.howstuffworks.com/cameras-photography/digital/digital-camera.htm (visited on 11/03/2016).

[56] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, 2004. ISBN: 0521540518.

[57] Corinna Jacobs and I Parrish. *Interactive Panoramas: Techniques for Digital Panoramic Photography*. Springer Science & Business Media, 2012. ISBN: 9783642186653.

[58] Robert Schowengerdt. *Remote Sensing: Models and Methods for Image Processing*. Elsevier Science, 2012. ISBN: 9780080516103.

[59] S.A. Quadri. *What is Spatial Resolution?* URL: http://www.slideshare.net/reachquadri/what-is-spatial-resolution (visited on 11/02/2016).

[60] Evan Mclean Smith. "A Collection of Computer Vision Algorithms Capable of Detecting Linear Infrastructure for the Purpose of UAV Control A Collection of Computer Vision Algorithms Capable of Detecting Linear Infrastructure for the Purpose of UAV Control". PhD thesis. 2016.

[61] David Fleet et al. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part 7*. Springer International Publishing, 2014. ISBN: 9783319105840.

[62] *Key Differences Between Rolling Shutter and Frame (Global) Shutter - Point Grey Knowledge Base*. URL: https://www.ptgrey.com/KB/10028 (visited on 11/02/2016).

[63] *RGB Color Space - Wikipedia*. URL: https://en.wikipedia.org/wiki/RGB%7B%5C_%7Dcolor%7B%5C_%7Dspace (visited on 10/30/2016).

[64] *RGB Color Model - Wikipedia*. URL: https://en.wikipedia.org/wiki/RGB%7B%5C_%7Dcolor%7B%5C_%7Dmodel (visited on 10/30/2016).

[65] *HSL and HSV - Wikipedia*. URL: https://en.wikipedia.org/wiki/HSL%7B%5C_%7Dand%7B%5C_%7DHSV (visited on 10/30/2016).

[66] Alvy Ray Smith. "Color gamut transform pairs". In: *ACM SIGGRAPH Computer Graphics* 12.3 (1978), pp. 12–19. ISSN: 00978930. DOI: 10.1145/965139.807361.

[67] Ramesh Jain, Rangachar Kasturi, and Brian Schunck. *Machine Vision*. McGraw-Hill, 1995. ISBN: 9780070320185.

[68] Devi Parikh. *Linear Filters [PowerPoint Presentation] - ECE 5554: Computer Vision*. 2015. URL: https://filebox.ece.vt.edu/%7B~%7DF15ECE5554ECE4984/lectures/parikh%7B%5C_%7Dlecture2%7B%5C_%7Dfilters.pptx (visited on 08/27/2015).

[69] *Binary Image - Wikipedia*. URL: https://en.wikipedia.org/wiki/Binary%7B%5C_%7Dimage (visited on 11/03/2016).

[70] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley & Sons, 2011. ISBN: 9781119957003.

[71] Joseph Wilson and Gerhard Ritter. *Handbook of Computer Vision Algorithms in Image Algebra*. Second. CRC Press, 2000. ISBN: 9781420042382.

[72] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 3400–3407. URL: http://april.eecs.umich.edu.

[73] Sanni Siltanen. "Theory and applications of marker-based augmented reality". PhD thesis. 2012.

[74] *ARToolKit Website*. URL: http://www.hitl.washington.edu/artoolkit/ (visited on 10/29/2016).

[75] *ALVAR Toolkit Website*. URL: http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html (visited on 10/29/2016).

[76] *AR Track Alvar - ROS Wiki*. URL: http://wiki.ros.org/ar%7B%5C_%7Dtrack%7B%5C_%7Dalvar (visited on 10/29/2016).

[77] Thomas Moore and Daniel Stouch. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, 2014.

[78] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. ISSN: 01628828.

[79] J. Illingworth and J. Kittler. "The Adaptive Hough Transform". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), pp. 690–698. ISSN: 01628828.

[80] Jianbo Shi and Carlo Tomasi. "Good Features To Track". In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference*. 1994, pp. 593–600. ISBN: 0-8186-5825-8. DOI: 10.1109/CVPR.1994.323794.

[81] Simon Baker and Iain Matthews. "Lucas-Kanade 20 Years On: A Unifying Framework". In: *International Journal of Computer Vision* 56.3 (2004), pp. 221–255.

[82] *Robot Localization - ROS Wiki*. URL: http://wiki.ros.org/robot%7B%5C_%7Dlocalization (visited on 11/06/2016).

[83] *AR Track ALVAR - ROS Wiki*. URL: http://wiki.ros.org/ar%7B%5C_%7Dtrack%7B%5C_%7Dalvar (visited on 11/06/2016).

[84] Itseez. *Open Source Computer Vision Library*. 2015. URL: https://github.com/itseez/opencv.

[85] Thiemo Wiedemeyer. *IAI Kinect2*. \url{https://github.com/code-iai/iai_kinect2}. University Bremen.

[86] Jean-Yves Bouguet. *Camera Calibration Toolbox For Matlab*. 2015. URL: https://www.vision.caltech.edu/bouguetj/calib%7B%5C_%7Ddoc/index.html (visited on 11/06/2016).

[87] Zhengyou Zhang Zhengyou Zhang. "Flexible camera calibration by viewing a plane from unknown orientations". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 1. c. 1999, pp. –7. ISBN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.791289.