Location and Mapping for NASA Sample Return Robot Challenge

Submitted in Partial Fulfillment
of the Requirements for SYDE 462

Caleb Gingrich, 20269791, 4B
Daniel Johnson, 20270969, 4B
Mohan Thomas, 20288204, 4B

Faculty of Engineering
Department of Systems Design Engineering

April 2, 2012.
Course Instructor: Professor M. Gorbet

Executive Summary

The goal of this project was to develop a simultaneous localization and mapping (SLAM) system for an autonomous robot competing in the NASA Sample Return Robot Challenge. This challenge asks for a robot to explore a large amount of outdoor terrain, collect a number of samples of interest, and then return to its starting location all without any human intervention. It's important for the robot to track it's location accurately throughout the competition to be able to plan future routes and to enable the robot to return home at the end of the time limit. The robot will be equiped with an IMU, wheel odometry, and laser radar scaners for the purposes of locaiton and mapping.

A study of the current state of the art found that very few techniques exist for estimating the location of a robot in a 3-dimensional (6 degree of freedom) environment in real time. While a lot of work has been done in reconstructing 3-dimensional terrain and robot paths in post-processing, the state of the art for real-time 6DOF SLAM is limited. Current techniques use Kalman Filters and visual odometry, wheel odometry, and IMU measurements to estimate the robot's location. In post processing, scan-matching and global optimization approaches applied primarily to laser radar scans have been used.

The proposed solution combines a Kalman filter with scan-matching using Iterative Closest Point (ICP) to obtain real-time estimates of the robot's location. In the background, a global optimization process using GraphSLAM is performed to further refine those estimates and when it completes, the refinements are passed on to any future pose estimates made.

Using this strategy, some good results have been achieved. The estimates from the proposed algorithm provide smaller error in localization while the robot travels through the simulation environment. Nevertheless, there are still some issues to be resolved. Improvements could be made to the mathematical and computational robustness of the algorithm, as well as other components considered within the same general framework.

This report shows that combining these techniques is a viable option for performing real-time 6DOF SLAM while recognizing that this is a difficult problem which still requires further work before the system can be used on the competition robot.

Contents

List of Tables

List of Figures

1

Background

The goal of this project is to design and build the mapping and location algorithms for the NASA Sample Return Robot Challenge (NSRRC). The overall competition goal is to build a robot or team of robots that can autonomously explore $80\,000\,\mathrm{m}^2$ of terrain, recognize samples of interest, retrieve and store the samples within the robot, and finally return the robot to its starting location [2]. The challenge was created by NASA to simulate a lander sent to the moon or Mars that would collect samples to be returned to Earth. A team was created at the University of Waterloo to enter this competition. As part of this team, this project is specifically focused on the selection of sensors and the development of software algorithms that will allow the robot to determine its location and map its surroundings, performing SLAM for the competition.

The goal of our team was to design the location and mapping algorithms needed for the project. The output of the algorithms will be a set of poses that indicate where the robot has travelled. These poses and the accompanying map will be used by the route-planning teams for global and local planning. The provided information needs to be accurate and precise as the map and robot pose information will be used for directing the robot to samples and for avoiding hazards and obstacles. Maintaining the correct state for the roll, pitch, and yaw and position of the robot is essential for correct control of the robot drive motors. Since the robot needs to know it's location at all points during the competition, the algorithms developed must run in real time.

This chapter outlines the required background information for understanding the design

problem and the solution presented in this report. Section 1.1 outlines the general challenge of robotic localization in the context of the NSRRC and Section 1.3 explains more specifics of the algorithms that were used as part of the final design solution.

1.1 Robotic Mapping in the Context of the NSRRC

Robotic mapping, localization, and planning are important problems in the design of autonomous robots relying on sensor data to make decisions about navigation in a foreign environment. Thrun's Robotic Mapping: A Survey covers the problem comprehensively [3]. Some of the challenges highlighted include:

- High dimensionality in the representation of a map from sensor data
- Nonlinearities in environment and intrinsic noise
- Correspondence problem: the difficulty associated with recognizing two measurements of the same object in a map, perhaps from different perspectives
- Dynamic or changing environments
- Exploration given incomplete and possibly incorrect environment maps

Unique challenges are presented in the NASA Sample Return Robot Challenge. An outdoor environment will be used in the competition to simulate a martian or lunar environment. Prominent landmarks may be sparse, and terrain may be highly heterogenous and potentially un-drivable in some locations.

Simultaneous Location And Mapping (SLAM) is a general term for any technique enabling a robot equipped with spatial sensors to construct a map of its environment as it is traversed while concurrently estimating the robot's pose (position and orientation) within the environment. A common technique is to integrate odometry measurements of the robot movement with acceleration motion from an inertial measurement sensor using Bayesian filters (such as the Kalman filter) to determine position estimates [4]. This approach tends to accrue error over time as each new estimate is based on the last.

Location and mapping in sparse outdoor terrain is a challenge for any approaches founded on assumptions which hold in indoor environments. Six-degrees-of-freedom SLAM (6DOF SLAM) accounts for all spatial information - $(x, y, z, \theta_x, \theta_y, \theta_z)$ - in the generation of maps and pose estimates. The use of 6DOF SLAM has been demonstrated for outdoor environments by several groups, but most of the work has been for offline experiments, building maps after data has been acquired, rather than real-time SLAM. For example, Nüchter et al. used detailed 3D laser range derived point clouds in 6D SLAM to determine drivable paths in a foreign environment [5].

In general, localization using 3D laser scans involves solving a problem which is referred to as scan registration or scan matching. The goal of scan matching is to determine the motion of the robot by examining successive scans of the environment, finding common features and transforming the scans into the same global frame of reference. The robot's motion can then be infered from the relative positions of the two scans. The Iterative Closest Point method (ICP) [6] is a well known technique for registration of three dimensional scans. Genetic implementations improving its convergence have also been demonstrated [7, 8]. However, ICP is significantly more challenging with 6DOF [9]. Other methods for scan registration have also been developed [10]. Because of the sheer magnitude of data associated with full laser scans, scan reduction is often performed before matches are performed. Despite its improved accuracy, scan matching techniques still tend to accrue error over time as each pose estimate is still based on the last.

Finally, in post-processing implementations, global optimization approaches have been used to solve the SLAM problem. In general, these work by recognizing when the robot has returned to a location multiple times (loop detection) or seen the same feature multiple times and then shifting the previous pose estimates so that the locations of static features in the environment are consistent. GraphSLAM works by constructing a linear graph of constraints that is then optimzed [11]. Unfortunately, the computational complexity of most optimization SLAM implementations increases significantly for additional degrees of freedom. Nüchter et al made use of heuristic closed loop detection in laser range data for optimized 6DOF SLAM [4]. Global optimization approaches are able to spread the error accrued over time over all pose estimates and minimize the error of the last pose measured.

These approaches have not been successfully used in a real-time SLAM algorithm as they are too computationally intense and the complexity increases with time.

Further work will need to be done to determine effective ways of integrating information from multiple on-board sensors, including laser and vision systems, and also odometry and inertial measurements. Jeong et al [9] made use of both vision and laser range data. Asmar presented a vision-inertial based slam technique for outdoor environments [12].

1.2 Novelty and Utility

The development of an effective real time 6DOF SLAM algorithm capable of running in real time will enable the developed robot to autonomously navigate new terrain while keeping track of its location. Real-time localization is absolutely an integral component for any autonomous self-contained exporatory system. Quality of the information in localization and mapping data will greatly affect the choices of the path-planning algorithms used by the robot.

6D SLAM in irregular environments is still considered a challenge, and very few real-time implementations using laser range data are available, because the magnitude of data and the complexity associated with manipulating the data are significant challenges. Novel or innovative approaches to scan matching, sensor data fusion, parallel computation and data representation may be necessary in order to develop an effective algorithm.

1.3 Technical Background

Three separate algorithms that were use in this project require a deeper explanation to understand the design choices made. While these algorithms were not implemented as part of the project it is important to have a basic understanding of how they work to understand the final implementation. These three are the extended Kalman Filter (EKF), Iterative Closest Point (ICP), and GraphSLAM. Each of these represents a different way of estimating robot pose from the available sensor information.

Extended Kalman Filter

The EKF takes as its input multiple estimates of the robot motion and combines them probabilistically to obtain an improved estimate. In our case, odometry information based on the wheel spin of the robot, and acceleration and orientation data from an inertial measurement unit (IMU) is used to estimate the change in pose.

In general, Kalman filters update based on a two step process [1]. In the first step, previous knowledge of the robot's position along with a known model for the robot motion is used to provide a guess for the next state of the robot, x_k. The motion model (f) calculates the next state and its uncertainty based on an input, u_k, the previous state, x_{k-1}, and the uncertainty in the previous state w_{k-1}.

$$x_k = f(x_{k-1}, u_k, w_{k-1})$$

The predicted pose (x_k) is then used along with data from sensors to calculate the measurement update (z_k). This correction also includes some uncertainty in the measurement modeled by v_k.

$$z_k = h(x_k, v_k)$$

In a standard Kalman filter, the functions on the right hand side are linear, and the final pose estimate can be calculated by combining the estimate of the next state with the measurement estimate. The Kalman gain (a measure of the relative confidence of the estimate and the correction) is used to determine the weight given to the estimate and the correction in the final output. This output is then used as the input to the next calculation step. Figure 1.1 illustrates the process and details the equations used.

An EKF can use a non-linear motion model or measurement model but linearizes the equations of the motion and measurement models using a Taylor Series approximation at each time step about the previous pose estimate. The same linear process can then be used to estimate the next robot state.

An EKF is able to maintain a better pose estimate than either the odometry information

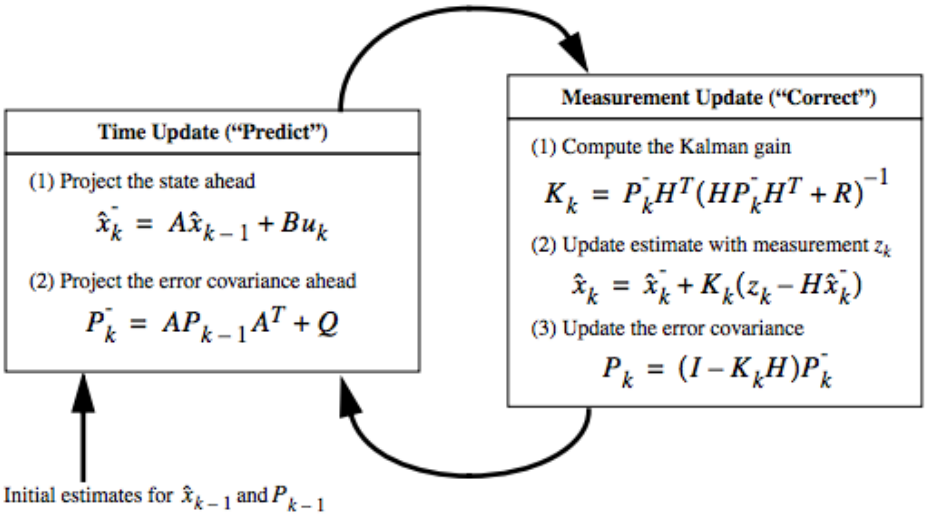


Figure 1.1: Linear Kalman Filter Algorithm [1]

or the integration of IMU measurements on its own. It is computationally efficient and does not require expensive or complicated sensors. Nevertheless it is not suitable on its own as a localization solution. At each time step, error from the previous step is preserved and the EFK tends to drift over time away from the robot's true location. Error accrues as time passes and after multiple hours the pose estimates are inaccurate. Over short distances the EKF is able to measure the robot's pose accurately but it cannot be used for longer paths.

An implementation of an EKF is available within ROS as the node robot_pose_ekf. With minor modifications, this node was used within the final design.

Iterative Closest Point

The Iterative Closest Point algorithm is a method for comparing two 3-D point clouds and determining the difference between the poses at which they were taken. A point cloud is a measurement of the robot's surroundings that shows the distance to the nearest obstacle in all directions. A sample point cloud from a simulation environment is shown in Figure 1.2. In this cloud, several trees are visible along with the ground plane. ICP takes as input two point clouds, the pose of the first point cloud, and an estimate for the pose of the second. It outputs an improved estimate for the pose of the second point cloud. ICP can be used to determine the motion of the robot by comparing two point clouds that come from

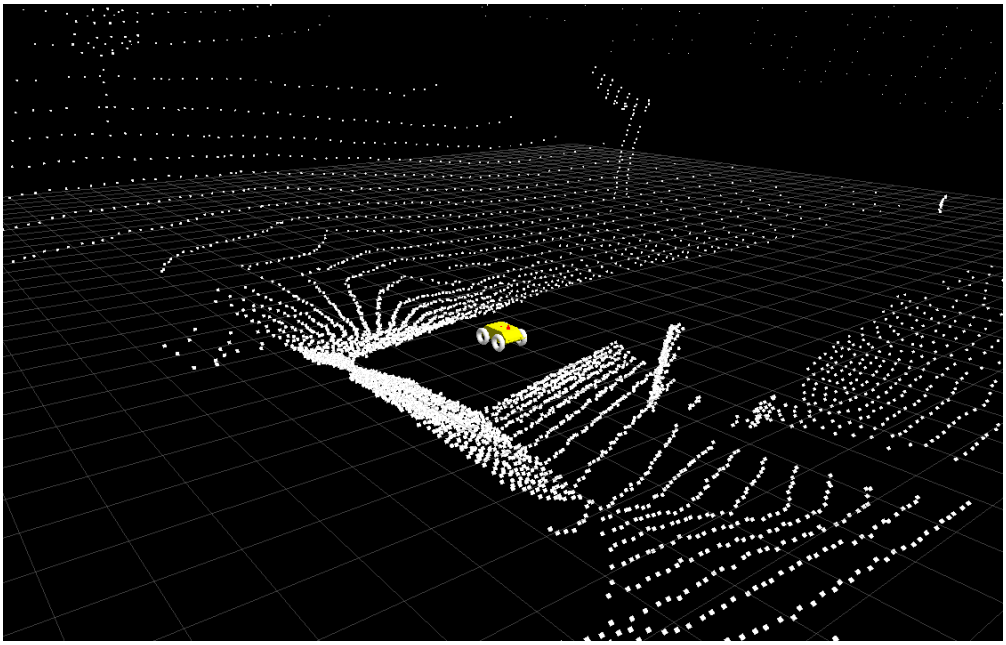


Figure 1.2: A Sample Point Cloud from Simulation

consecutive laser scans of the robot's environment [13]. In contrast to the EKF Filter, ICP is an optimization method, minimizing a error function rather than filtering estimations with uncertainty models.

ICP examines corresponding points in the two point clouds and figures out the transformation (i.e. rotation and translation) that minimizes the sum of the differences in position between them (the error function). This transformation is representative of the robot's movement during the time between the scans were taken. This is analogous to the idea that a person could examine their surroundings in one location, be transported somewhere nearby, and then determine their new location based on the new positions of features around them which they observed in the first location. For people, this is not a particularly difficult task as it's easy to recognize corresponding objects visible from both locations. For the robot, determining correspondence is more difficult.

Instead of extracting features from the point cloud and trying to match their locations, ICP depends upon an initial estimate that is used to transform the second point cloud into the same coordinate frame as the first. Then, points that are closest to each other are assumed to correspond and the transformation calculated. This new transformation is applied to the second point cloud, the correspondences are recalculated, and the process is

repeated iteratively until either a maximum number of iterations has been reached or the points no longer move in the next iteration.

Mathematically, ICP minimizes the error function by solving for the the rotation and translation $(\mathbf{R}, \mathbf{t})$ of the second scan [13]:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} ||\mathbf{m_i} - (\mathbf{R}\mathbf{d_j} + \mathbf{t})||^2 \tag{1}$$

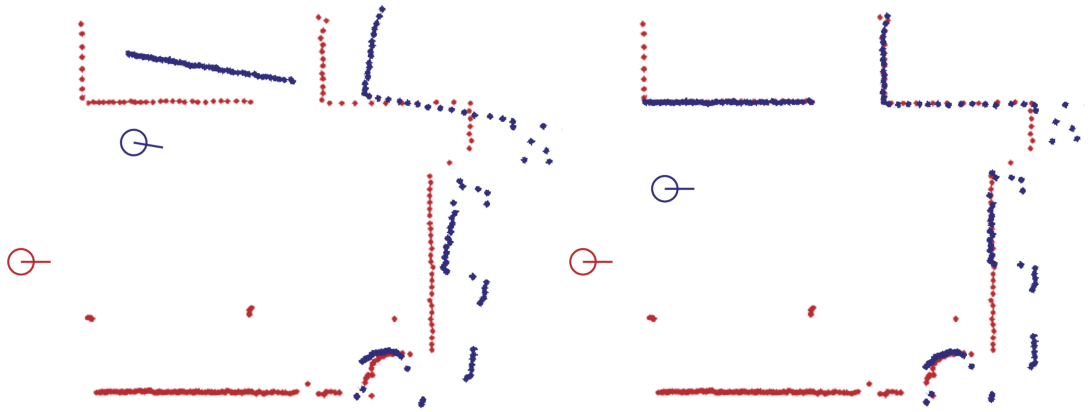
where N_m is the number of points in the first scan (the model set, M), N_d is the number of points in the second scan (the data set, D). The weighting factor ($w_{i,j}$) is used to set the correspondence of points and is set to 1 if $\mathbf{m_i}$ is the closest point to $\mathbf{d_j}$ and 0 otherwise. For partially overlapping point scans a maximum distance is set for corresponding points above which $w_{i,j}$ is always 0. This maximum distance is an important parameter for the algorithm that needs to be optimized for any particular environment. This equation reduces to:

$$E(\mathbf{R}, \mathbf{t}) \propto \frac{1}{N} \sum_{i=1}^{N} w_{i,j} ||\mathbf{m_i} - (\mathbf{R}\mathbf{d_i} + \mathbf{t})||^2 \tag{2}$$

by only including the points which have a corresponding point in the other scan with $N = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j}$. At each iteration of ICP this equation can be solved using a number of different methods including SVD decomposition, quaternions, orthonormal matrices, dual quaternions, or a helical method [14]. These methods give similar performance given noisy data [13]. ICP assumes that by the last iteration, we will have achieved close to perfect point correspondence across the two scans.

Figure 1.3 helps to illustrate the algorithm. On the left, we include the model scan in red and the data scan in blue along with the estimated pose for where the robot was. Through ICP, the rotation and translation that needs to be applied to the data scan is determined that lines it up with the model scan as shown on the right. This same transform is then applied to the blue robot pose and the estimate for the robot's motion is the difference between the red pose and the corrected blue pose.

ICP includes information from only the last two scans and is susceptible to the same drifting problems as the Kalman Filter albeit at a slower pace as there is less error in each



(a) ICP Initial Estimate (b) ICP Final Alignment

Figure 1.3: Aligned Scans after ICP

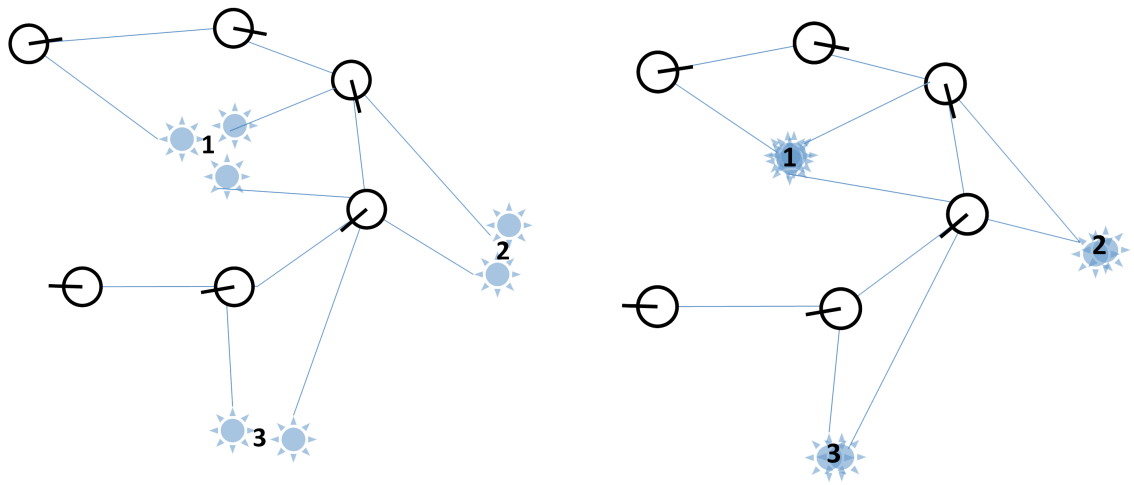
estimate. ICP is still relatively fast and can be performed on smaller point clouds in real time.

A number of different ICP implementations are available in the 3D-Toolkit distributed by Jacobs University and the University of Osnabrück [15]. This library provides opportunities to test a variety of ICP implementations and compare their performance within our localization framework.

GraphSLAM

GraphSLAM is an optimization technique that uses the entire history of laser scans and pose estimates to perform localization. Instead of using information only from the last two scans, GraphSLAM uses the entire history of pose estimates and adjusts them based on constraints developed by a graph of poses and features. Figure 1.4 illustrates the constraints developed in a simplified robot path.

In this simple path of 6 estimated poses, 3 different features are observed, some of them from multiple poses. For example, feature 1 is observed from the 1st, 3rd, and 4th position. Assuming a static environment, the estimated position of feature 1 should be the same regardless of the poses it was observed from. But, because of errors in the pose estimates and the measurements of the feature, we can see in figure 1.4a that the estimates for the position of feature 1 are not the same. The goal of GraphSLAM is to minimize these errors



(a) GraphSLAM Initial Pose Estimates (b) GraphSLAM Final Pose Estimates

Figure 1.4: GraphSLAM

globally throughout the entire graph of poses and features by shifting the poses. In the graph there are also measurements obtained from the motion model for the robot (e.g. the difference between the 1st and 2nd pose in the graph). These measurements are also included as constraints in the graph and their error included in the minimization problem [11].

Since all of the measurements are probabilistic with assumed Gaussian noise, the minimization problem is converted to a maximum likelihood formulation where the goal is to maximize the likelihood of the state of the system over all time ($z_{0:t}$) given the motion model ($u_{1:t}$), the measurements ($y_{1:t}$), and the correspondence of features ($c_{1:t}$).

$$\max_{z_{0:t}} p(z_{0:t}|y_{1:t}, u_{1:t}, c_{1:t}) \tag{3}$$

This problem can be reformulated using Bayes Theorem and taking the negative log likelihood to become an unconstrained non-linear minimization.

$$\begin{aligned}\min_{z_{0:t}} J = K + [x_0 - \mu_0]^T \Sigma_0^{-1} [x_0 - \mu_0] + \sum_t [x_t - g(x_{t-1}, u_{t-1}]^T R^{-1} [x_t - g(x_{t-1}, u_{t-1}] \\ + \sum_t \sum_i [y_t^i - h(z_t, c_t^i)]^T Q^{-1} [y_t^i - h(z_t, c_t^i)]\end{aligned} \tag{4}$$

Where z_t indicates the state of the system at each time step, x_t is the robot pose, g is the motion model given the previous state and the robot control input u, h is the measurement

model, μ is initial pose, and Σ, R, and Q are the covariance uncertainties associated with each step. Solving this minimization is complicated and a number of methods exist for doing so [11]. Thrun details one possible solution [16].

GraphSLAM can obtain better estimates for the robot pose and since the error is spread throughout the graph, the problems of drifting pose estimates over time are minimized [16]. Unfortunately, this global optimization problem takes significant computation resources to solve and likely cannot be used on its own as a real-time algorithm.

The 3D-Toolkit also includes a number of different implementations of GraphSLAM [15]. Like ICP, there a number of different ways of solving the optimization problem and using the 3D-Toolkit allows for efficient comparison of the various techniques.

2

Engineering Design

This chapter presents the details of the engineering design process as used in this project. Section 2.1 identifies the design requirements, as this was the first stage. The required scope refinements are discussed, and the final design requirements are presented. Section 2.3 presents the identified solutions, and discusses the selection process. Section 2.4 presents details of the final solution. The significant technical challenges encountered and their solutions are discussed in Section 2.5.

2.1 Design Requirements

Design Objective

The goal of this project is to design a real-time location and mapping algorithm which will allow a robot to determine its location with respect to observed surroundings, using the on board computational resources, and sensors such as LiDAR, cameras, an Inertial Measurement Unit (IMU) and wheel odometry measurements. As little prior information of the surroundings will be provided, this is a typical SLAM problem.

Development and Revision of Objectives

The scope of the project has changed over the course of the past two terms. Initially, mapping, location and planning were all considered objectives of the project, but it quickly

became clear that planning in itself is a very complex problem. As such, two independent sub-teams in the competition group are working on planning at the global and local level. The global level is how to explore a territory while managing time and power, while the local level determines how to navigate around obstacles and to a sample. Accordingly, our team's scope was reduced to location and mapping.

The original goal for the end of term was to have a mapping and location algorithm tested, tuned and validated with the actual robot and sensor being used. Unfortunately, the sensor was only available very late in the winter term. Therefore, this project focuses on the modular development of location and mapping algorithms using ROS (Robot Operating System). The rich simulation environment (created in the Fall term) provides a suitable platform for algorithm validation. ROS is the same platform the actual robot will be using, thereby minimizing integration difficulties. The actual integration effort will begin shortly after the completion of the academic term.

'Location' refers to real-time current pose estimation. 'Mapping' refers to the maintenance of a representation of the explored territory. In the simplest form, mapping may consist of applying pose estimates as transforms to corresponding laser scans. The map is then simply the combined point cloud from the laser scans. More involved implementations include map representation using sophisticated data structures appropriate for quick traversal and collision checking. The format of maps for this project is the former, simpler case. More complex structures may become necessary during the competition integration phase.

Functional Requirements

The functional requirements of this project are the tasks which need to be performed by the robot to determine its location and build up a map of its surroundings. A list of tasks is included below.

1. Perform scans of the surrounding area.

2. Perform pre-processing on incoming scan data (to reduce the effects of noise, for example).

3. Determine the pose (location and orientation) of the robot in real time as the robot travels through the environment.

4. Build up a map of the surroundings as the robot moves through the environment.

The final output will be a set of three dimensional (6 degrees of freedom) poses indicating the trajectory of travel over time. These poses will be used by the planning teams for global and local planning. In addition, a map may be constructed using these pose transforms and the original scan data. The information provided needs to be accurate and timely as the map and robot pose information will be used for directing the robot to samples and for avoiding hazards and obstacles. Maintaining the correct state for the roll, pitch, and yaw and position of the robot is essential for correct planning, and consequently for correct control of the robot drive motors.

Design Constraints

Competition Rules

The design which we will use is restricted first by NASA Sample Return Competition rules published in September 2011 [2]. For the location and mapping portions of the competition, the relevant rules their corresponding implications are:

- Any sensors relying on magnetic fields or sound-based signals are not permitted: may not use magnetometers
- Global Satellite Positioning Systems (GPS) or other satellite or radio based location systems may not be leveraged.
- The entire robot, including all sensing and computing equipment, may not exceed a mass of 80 kg, and a size of 1.5 m in each dimension
- Computational resources will be restricted to an on board laptop.
- The robot's traveling speed should be no more than $2\,\mathrm{m\,s^{-1}}$: any location algorithms will need to be fast enough to keep up with the robot speed

The competition rules greatly reduce the set of possible sensors which can be used for mapping and location. We are restricted to using laser sensors (LiDAR - Light Detection And Ranging), cameras, Inertial Measurement Units (IMUs) and odometry information from wheels in our design.

Sensor and Hardware Selection

Following discussion with the competition team regarding competition budget and available hardware, specific implications for our design team include:

- Mapping and Location algorithms must run in real-time on one high-performance laptop
- Available Sensor: Velodyne HDL-32E LiDAR
- Available Sensor: Wheel odometry
- Available Sensor: 3DM-GX3 -25 IMU (without tri-axial magnetometer information)
- Available Sensor: Cameras

Laser range data is most appropriate for determining spatial information about the surrounding environment. The Velodyne LiDAR listed above was acquired by the competition team. It is a sophisticated sensor which generates dense point clouds representing the spatial characteristics of the surrounding environment within a 100 m radius. For estimation of distance, LiDAR is much more accurate that vision-based systems (such as cameras) for anything beyond a few meters [17]. The competition team was able to acquire a Velodyne HDL-32E laser range scanner, a sophisticated LiDAR sensor which which our group plans to use.

2.2 Final Scoped Problem

A technique must be developed to generate the following outputs:

- *3-Dimensional Location* - Real Time 6-DOF pose estimates must be generated at least once every two seconds. (Primary Goal)
- *3-Dimensional Mapping* - The pose estimates and original scan data will be used to represent 3-dimensional terrain. (Secondary Goal)

The algorithm must run within these constraints on a single laptop.

2.3 Initial Design Concept

Background research revealed several implementations of 6D SLAM in outdoor environments [18][5][19]. Nearly all of these implementations were developed as post-processing procedures. That is, given a collection of scans of some environment, the 3D environment is reconstructed by aligning scans with 6 degrees of freedom. In this case, pose estimates are a secondary output - the primary output is the reconstructed map. Most of these techniques made use of laser range scans as inputs. Vision was only used in outdoor SLAM as a supplement to LiDAR sensing, or as a cheaper alternative appropriate for environments with salient landmarks [12].

In general, a technique is necessary for processing raw laser scans to determine the spatial relationship or relative orientation between them. This needs to be done in real time. The environment may be scanned at a frequency ranging from 10Hz to 0.5Hz. Each scan should have a corresponding pose estimate.

Prior Attempts

Several techniques exist. Each technique has its advantages and disadvantages, as discussed in the technical background. A brief summary follows, ordered by ascending computational complexity, existing alternatives include:

- *Odometry and IMU integration* (e.g. using an Extended Kalman Filter - see background): Odometry and IMU data are directly correlated with the physical motion of the robot, and estimating position using both is fairly straightforward, and may be

accomplished using an Extended Kalman Filter. However, both sensors are extremely sensitive to noise, and error accumulates for subsequent position estimates.

- *Scan Matching*: Motion between subsequent scans is inferred based on spatial observations. This technique is less susceptible to corrupting noise when compared to odometry-based estimation as multiple independent 'observations' in each scan are used. However, error accumulation and the subsequent estimation drift is still a problem. 3D Scan matching is possible in real time, but more computationally intensive than odometry-based techniques. Some scan matching techniques, such as ICP (Iterative Closest Point), are very sensitive to the initial guess for relative position and orientation between scans.

- *GraphSLAM - Global optimization of scan positions*: GraphSLAM distributes error globally by finding the relationship between all scans at once. When using 3 Dimensional data, computational performance is a significant concern for any appreciable number of scans. Traditionally 6DOF SLAM is not a real-time procedure, but it is much more accurate than either of the previous techniques.

GraphSLAM is the ideal solution for accuracy - in fact, in many real-time 2D location and mapping problems, 2D GraphSLAM is used [3]. Requiring 6-DOF pose estimates from 3D scan data adds significant complexity, however, and scaling real-time GraphSLAM to the 3D space is far from trivial. Odometry and inertial estimates are useful, but only over short distances. Scan matching might seem like a reasonable compromise, but the potential impact of drift error is far too significant for this application: the robot must be able to return to its starting position after hours of autonomous exploration.

A common refinement is the use of odometry and IMU information between subsequent scans as an initial guess for ICP. This decreases the likelihood that ICP will erroneously converge on an inappropriate local minimum but it does not solve the problem of accumulated error over time.

Proposed Solutions

Ways of integrating or adapting these different techniques for a more robust real-time location (and mapping) solution are explored below:

1. Run ICP when the robot is moving, run GraphSLAM when the robot is stopped (e.g. when collecting a sample, or at intentionally scheduled 'GraphSLAM' stops). This technique is a good way of integrating both techniques, but requires irregular robot motion.

2. Run GraphSLAM on a small window of the last n scans. This approach attempts to align small groups of scans internally as best as possible. The last scan from one window is used as the first scan of the next run. The problem with the technique is the necessary delay before getting an up-to-date pose estimate - planning algorithms will need to extrapolate position while waiting for a new pose. In addition, it is not yet clear how much better this approach will be compared to ICP, if it is better at all. This technique is similar to scan matching, but using groups of scans rather than pairs.

3. ICP is run with every incoming scan, and GraphSLAM runs in the background on a parallel thread with lower priority. Each time GraphSLAM completes optimizing a set of scans and pose estimates, the improvement in the estimate is applied to any future run of ICP. This scheme allows for timely estimates to be provided while still gaining a benefit from global optimization and reducing overall drift.

The third alternative was selected for further development for a few reasons. The first alternative introduces restrictions on the motion of the robot, requiring the robot to stop moving before revised estimates can be made. The second alternative is a lower frequency and possibly higher quality alternative to scan matching, but error drift (while less that that for scan matching) is still present. In addition, the need for frequent up-to-date pose estimates may not be satisfied with this alternative. The third alternative should operate at a comparable speed to simple scan matching, but with periodic refinements. The first two alternatives may be appropriate in other applications, but the third option appears to be

the most promising solution for the NSRRC challenge. Figure 2.1 shows the system diagram of the selected design.

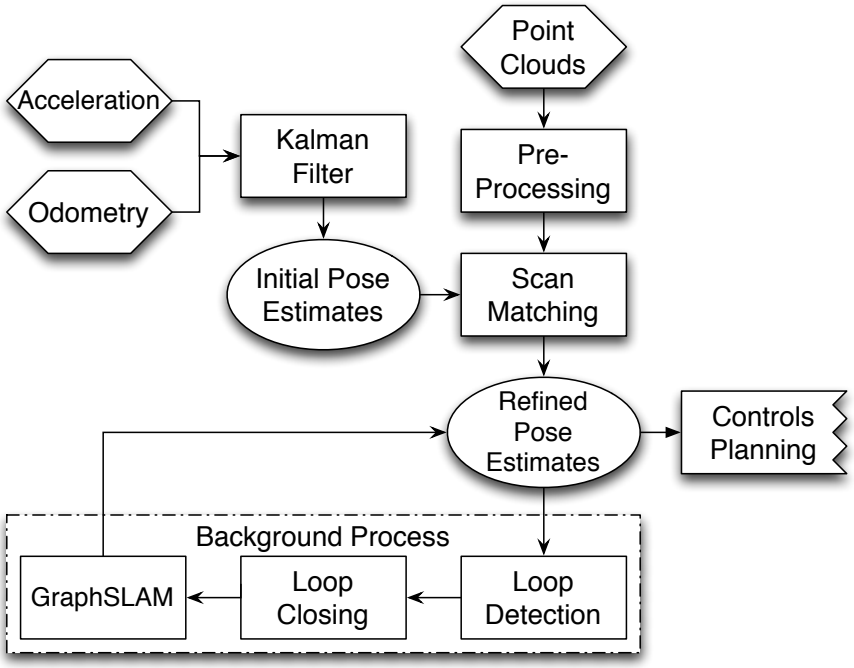


Figure 2.1: ICP with SLAM parallelized implementation

2.4 System Architecture

Existing algorithms for running SLAM and ICP offline are available. 3D-Toolkit is an excellent example of this [15]. This is a toolbox containing several implementations of ICP and SLAM, primarily design for running offline on large data sets. These tools were used to create the solution outlined in the last section.

At a high level, a new ROS node was developed to accept odometry estimates from an EKF filter and laser scan point clouds. These estimates are used to initialize scan matching. The node is responsible for running both GraphSLAM and ICP scan matching algorithms, and it outputs a pose estimate at least as quickly as the incoming scan frequency. Adapted versions of GraphSLAM and ICP from 3DToolkit are used by the algorithm [15]. Two threads are used by the ROS node, the main thread receives point clouds and uses scan matching to output a pose estimate while a background thread provides refined pose estimates from

GraphSLAM after a delay.

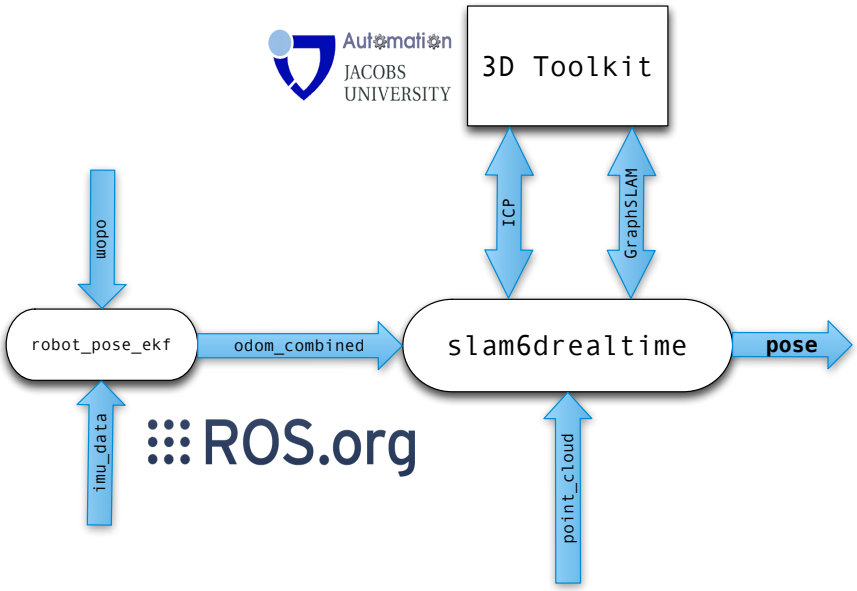


Figure 2.2: System interactions for the 'slam6drealtime' ROS node.

Scan and Pose Representation

Scans are stored using a SCAN class which contains the entire scan point cloud, a timestamp corresponding to the time of acquisition, and a 'transform'. This transform is a three-dimensional translation and rotation corresponding to the position of the scan relative to the starting position of the robot (at $t = 0$).

Along with each scan, a second transformation corresponding to the transformation from the position of the last scan to the position of the current scan is maintained. This is easily derived from the absolute pose of the two scans.

Scan Matching Thread

The ICP implementation from 3DToolkit was used for scan matching [15]. This thread maintains a history of the two most recent scans (currScan and lastScan), and estimates the relative transformation between them (diffTransform). A transformation consists of a three dimensional rotation and a three dimensional translation that corresponds to the change in robot pose during the time between the two scans. Once this transformation is calculated

Table 2.1: Scan Matching Variables

Variable Name	Type	Notes
currScan	Scan	Most recently acquired scan point cloud and corresponding pose estimate transform
lastScan	Scan	Second most recently acquired scan point cloud and corresponding pose estimate transform. Mutex protected.
diffTransform	StampedTransform	The transformation from lastScan to currScan
lastDiffTransform	StampedTransform	The transformation from the third most recent scan to lastScan
ICPDoneScans	deque	A vector-like object to which all scans prior to lastScan are added. Mutex protected for shared use.
ICPtransforms	deque	A vector-like object to which all scan-to-scan transforms prior to lastDiffTransform are added. Mutex protected for shared use.

and a new scan arrives, ICPDoneScans and ICP transforms, mutex-protected 'deque' objects, are updated with lastScan and lastDiffTransform respectively. These deque objects are also used by the SLAM thread. The scan matching thread was designed to run in real time with a scan frequency of 1 Hz (where a new scan arrives every second).

Initial Guess Strategy

As ICP is susceptible to convergence at local minima [7], the quality of initial guesses has significant impact on overall performance. The 'initial guess' must be some approximate relative transformation between currScan and lastScan. The pose estimate of lastScan is known. In order to roughly estimate the pose of currScan, an EKF filter is used to integrate odometry and IMU information and extrapolate from the lastScan pose. An simpler initial guess strategy would be to use the pose of the lastScan as an initial guess for the pose of currScan (assuming that the transformation between the scans is 0 translation and 0 rotation), but this strategy makes ICP more susceptible to convergence at local minima.

Table 2.2: SLAM Variables

Variable Name	Type	Notes
ICPDoneScans	deque	A vector-like object from which scans which have been processed by ICP are removed to be processed by GraphSLAM. Mutex protected for shared use.
ICPtransforms	deque	A vector-like object to which all scan-to-scan transforms for all scans which have not yet been processed by GraphSLAM are stored. Mutex protected for shared use.
SLAMScans	vector	A vector of all scans preceding the scans in ICP DoneScans. GraphSLAM is run on these scans

Global Optimization Thread

The algorithm selected for global optimization is GraphSLAM, discussed in some detail in the Technical Background. This thread is run with background priority on scans which have been processed by the scan matching thread. The purpose of this thread is to periodically improve pose estimation based on all scans, or a window of scans. The global optimization thread used all scans acquired from the start.

The GraphSLAM process is not time critical, and may take tens of seconds to complete. In general, the procedure looks at all scans and refines pose estimates in order to be globally consistent. Pose estimation by GraphSLAM is more robust than ICP only.

Update Strategy

As GraphSLAM runs on the SLAMScans vector, new scans and transforms accumulate in ICPDoneScans and ICPtransforms. The GraphSLAM thread only updates the poses of scans in the SLAMScans vector. Let lastSlamScan be the most recent scan in the SLAMScans vector, chronologically preceding the oldest scan in the ICPDoneScans vector. The update strategy for remaining scans is as follows:

- Starting from the oldest, set the pose of each scan in ICPDoneScans to the pose of the previous scan multiplied by the transform between the two from ICPtransforms. Remove the scan and transform from ICPDoneScans and ICPtransforms, and add the

scan to SLAMScans.

- For the oldest scan in ICPDoneScans, this means the pose is revised using the GraphSLAM refined pose of the most recent scan processed by SLAM (the newest pose in SLAMScans) and the transform between the two as established by ICP.

- For all following scans in ICPDoneScans, this change is simply propagated forward using subsequent transforms in ICPtransforms.

- Once the ICPDoneScans is empty, acquire the mutex lock on lastScan. This has the effect of briefly 'pausing' before ICP runs between lastScan and currScan. Update the pose of lastScan using lastDiffTransform. Release mutex lock.

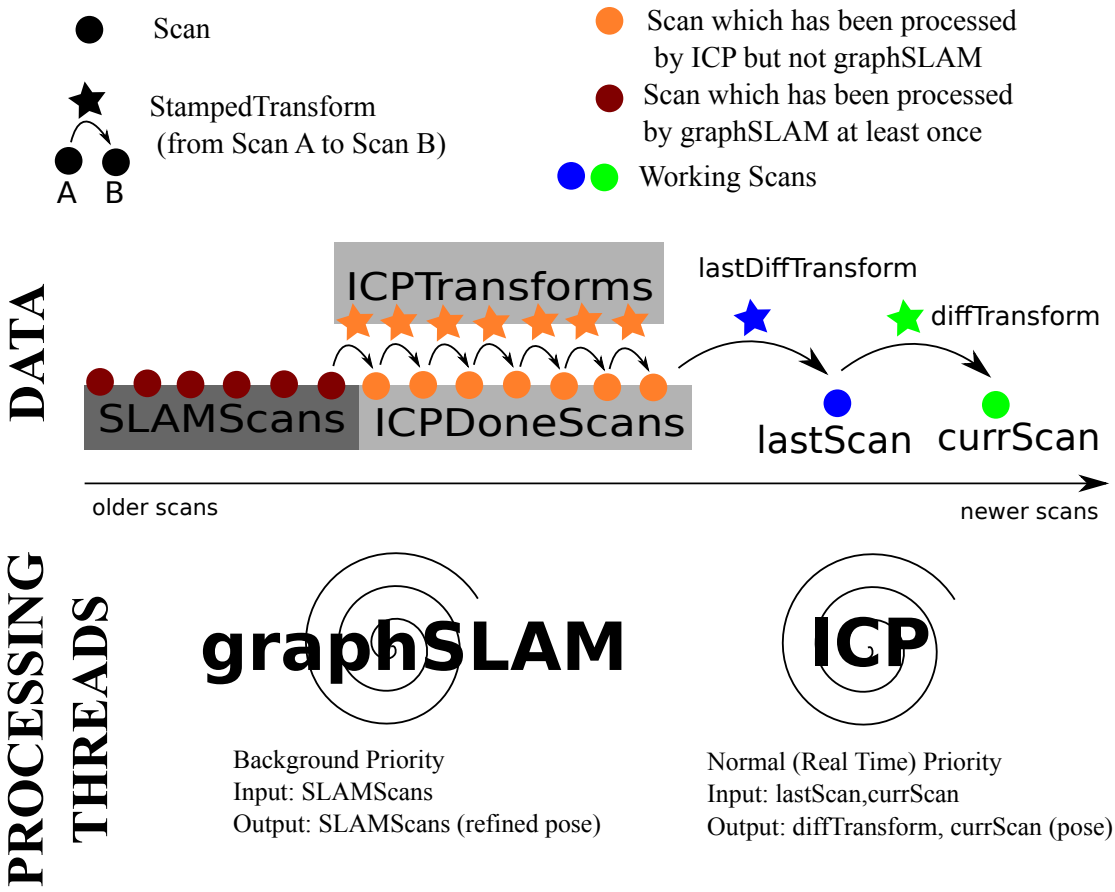
The effect of this procedure is that subsequent iterations of ICP incorporate the refinements to pose estimates from GraphSLAM. ICP continues to run in real time, providing timely pose estimates.

A vector of all of the pose estimates over the entire robot motion trajectory is made available as a final output (in addition to raw scan data and the current pose). The procedure is summarized in Figure 2.3.

2.5 Technical Challenges

One of the major technical challenges in this project was the lack of a platform for testing and validation. The physical robot was not ready for use early in the term, so it was necessary to develop a sophisticated environment and robot simulation for algorithm testing. This was a multi-faceted project in itself, requiring modeling of sensors, creation of a three dimensional terrain and the addition of tree-and rock-shaped obstacles.

CUDA (Compute Unified Device Architecture) is the name of a computing architecture which leverages Graphics Processing Units (GPUs) to perform complex parallelized operations efficiently. It was originally intended to develop using CUDA wherever possible, but this proved challenging because of firmware issues, and the amount of code refactoring required. This will likely be addressed before the competition date.



On completion of graphSLAM over SLAMScans, all subsequent scans up to lastScan need to be updated to include the refined estimate by SLAM

The pose of the oldest scan in ICPDoneScans is modified using the newest scan in SLAMScans and the corresponding Transformation. This procedure continues until lastScan's pose is updated.

In general, the transformation is applied thusly:

$$B = A*TF$$

where A is the older scan, B is the newer scan, and TF is the transform between the two determined by ICP. The propogation of the SLAM refinement is depicted below.

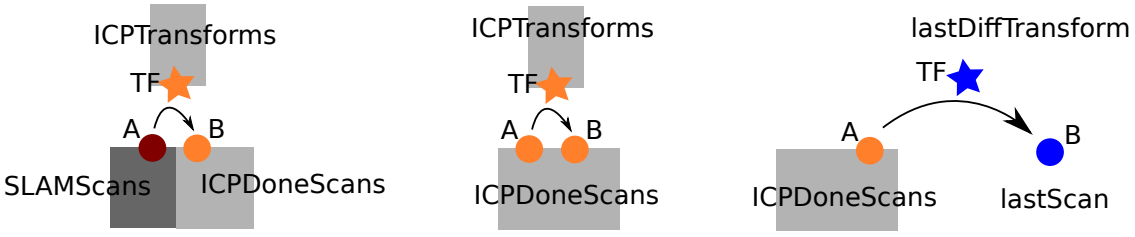


Figure 2.3: Illustration of interaction between GraphSLAM and ICP threads to provide timely and accurate pose estimates.

A significant challenge during implementation (rather than design) was adapting 3DToolkit and sample code provided by Prof. Andreas Nüchter. Among other things, a left-handed coordinate system was used in this code base (contrary to the conventional right-handed coordinate system). Learning to work with Robot Operating System (ROS) and develop on a simulation framework while developing with full competition team integration in mind was a challenge.

A further technical challenge which is anticipated in the weeks preceding the competition in June is integration and testing, validation and iteration using the actual robotic platform. New design constraints, such as power consumption, may become more significant.

3

Design, Testing and Evaluation

This chapter outlines how the design was tested to ensure that it meets the requirements. Section 3.1 describes the simulation environment that was created and how it was used to test the design. Section 3.2 outlines the attempts made to tune the parameters of the algorithm. Finally, section 3.3 shows the results of algorithm testing and outlines some outstanding issues which should be resolved.

3.1 Evaluation and Testing Methods

Testing and evaluation for this project consisted of both debugging software and evaluating the performance of the algorithm. This required the creation of several tools, discussed in the following subsection.

Tools

All evaluation and testing of the design was completed in the simulation environment constructed for this purpose last term. The was required as the hardware team had not finished constructing the robot and the required sensors were not available for use. The simulation environment also provided significant convenience and flexibility for testing. The dynamic simulation environment software Gazebo (included in ROS) was used, but maps of an outdoor environment including rolling terrain, irregular rock and trees were created by the

Figure 3.1: Simulation Environment

design team. The final result is depicted in Figure 3.1. The robot model was completed by other members of the competition team but a number of sensors were created by the design team. The terrain was simulated as a stationary trimesh, generated using Blender. A cloud texture was applied to a mesh, creating a smoothly varying but random colour map that is then used to adjust the elevation of each point [20]. The rocks were created by using QHull, a set of mesh creation tools, to find the convex shell that encloses a set of random points in the unit cube [21]. The trees were simply the rocks placed on cylinders of varying height and width. A Python script was used to construct a Gazebo model file for rocks and trees so the simulation world launches and runs as quickly as possible. The obstacles were randomly spaced out over the terrain, and placed at correct elevation. The script allows the dimensions of the terrain and the number of rocks and trees to be easily varied. The drawback of simulation is that the generated data can be perfect. Gaussian noise was added to the sensors used in the robot simulation as an attempt to capture their real-world performance.

Testing of the algorithm required capturing, storing and visualizing large amounts of data. A ROS node was constructed that could record the true pose of the robot at each scan, as well as the EKF estimates, ICP estimates and final refined pose estimates. This node

also included a small script to match the truth to the estimates. As the problem requires a real-time algorithm, timing information was required to evaluate the design. Timers were added to the code to report the time required to complete ICP, to deliver an initial pose estimate (including basic scan processing and pose updates as well as ICP), and to complete a SLAM run. Testing and understanding the operation of the code required seeing and comparing the accuracy of the estimates at many poses simultaneously in all six degrees of freedom. Two different visualization tools were created to see these results.

The path visualizer plotted the true path of the robot beside the EKF, ICP and SLAM estimations of the path in 3 dimensions [22, 23]. This script utilized the Python plotting library Matplotlib and the associated 3D plotting toolbox mplot3D. This tool was useful for seeing the overall behaviour of the algorithms, and displaying large errors in the estimates. This tool shows the position estimates well, but does not show the quality of the orientation estimates. It was useful for debugging the code, finding errors in the math and in the implementation of the math. An example of this visualization tool is shown in Figure 3.5.

An error visualizer was also created to show both the position and orientation errors over the whole path of the robot of several different estimates simultaneously. This was accomplished by plotting the true x and y coordinates of the robot at each scan on two different plots, one for position and one for orientation, also using the Matplotlib Python library [22]. At each scan in the path, the Euclidean distance from the truth to the estimate was plotted as the diameter of a circle centered on the point. This allowed seeing how the error was distributed over the entire path of the robot. The average, minimum and maximum error seen over the entire path for a given estimate were also reported numerically. An example of this visualization tool is shown in Figure 3.6.

Protocols

In order to test the computational performance of the implementation accurately, and to facilitate the comparison of testing results, the rosbag tool was used to capture a set of scans, true poses, and odometry estimates. This allowed the algorithm to be run on the same data quickly, without dedicating computational resources to the simulation at the same time. Several data sets of varying lengths with different paths through the simulation

environment were saved so the results were validated on a variety of conditions.

To test the design, different aspects of the algorithm were isolated and tested at different times. A process of rapid and frequent testing was used. With short data sets and effective visualization tools, changes could be made, testing and the results compared quickly. This allowed our team to understand and see the complex behaviour of many interacting components easily. Testing was completed on the same machine at all times, with no other programs running on the machine during the testing.

The testing began by validating the performance of the ICP algorithm in the new computational framework by disabling the SLAM process, and running the algorithm on a short data set. After visualizing the results, it was initially clear that the code needed to be debugged, as the ICP estimates were not consistently improvements over the odometry estimates. While this could also be caused by inadequate scan frequency and inappropriate selection of tuning parameters in the algorithm (requirements for point pair construction, minimization exit conditions), the defaults used by Nüchter in his code were used as good initial guesses. These variables were also varied and the results compared to each other. This allowed the impacts of these variables to be isolated from the other issues. These issues included small mathematical mistakes, and refining the process used to initialize ICP.

After the ICP algorithm was validated, the overall algorithm was run on the short data set. Initial bugs were found and removed in a similar visualize-change-visualize process. The algorithm was then tested on a longer data set that more accurately captures the required performance at the competition, and the results were saved and visualized using the same tools. The time required to complete each stage of the process was also checked.

3.2 Algorithm Parameter Tuning

Both the ICP and SLAM algorithms have several parameters that impact their performance. As this presents a very complex multi-variable optimization problem, especially because of the high sensitivity of the algorithms to environmental variables that are difficult to control, very little systematic optimization was possible. These environmental variables include amount of scan overlap, number of corresponding points, and true distance between

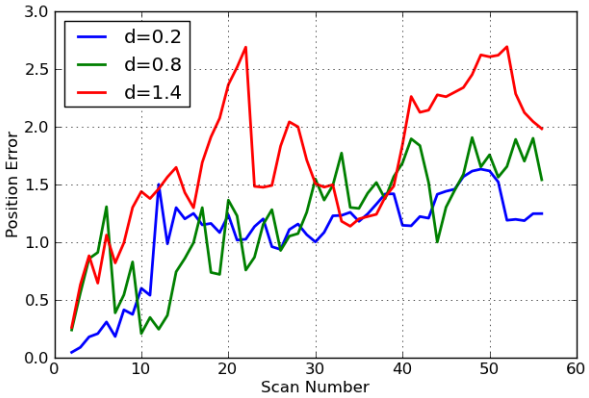
Table 3.1: Important Algorithm Parameters and their Tuned Values

Parameter	Final Value
Maximum number of ICP iterations	50
Maximum change in ICP error function between iterations	0.000 000 1 m
Maximum corresponding point distance for ICP	0.5 m
Maximum number of GraphSLAM iterations	20
Maximum change in GraphSLAM error function between iterations	0.001 m
Maximum corresponding point distance for GraphSLAM	0.5 m

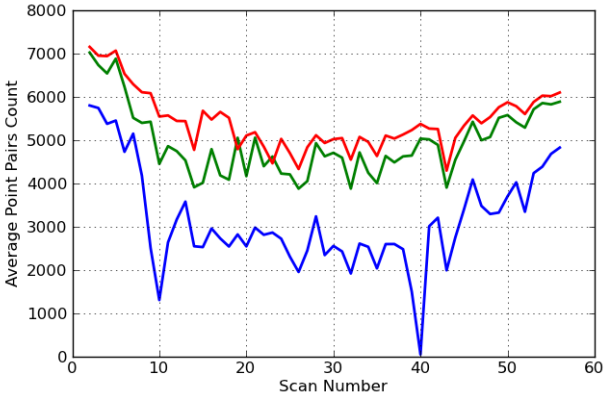
scans, all of which vary significantly over the path. As an inital cut at tuning, the important parameters were manipulated until the results looked reasonable. The important variables and their final values are listed in Table 3.1.

One variable was chosen to optimize, the maximum distance allowed between corresponding points of two scans (maximum corresponding point distance). The ICP algorithm was run on the same small data set many times with different values of this parameter. The estimate of the error, the run time per scan, the number of iterations per scan and the number of point pairs used for each scan was all collected. It was found that the maximum number of iterations and the convergence criteria for ICP have a significant impact. The convergence criteria recommended by Nüchter was used, and a large number of iterations was allowed. The error results are shown in Figure 3.4.

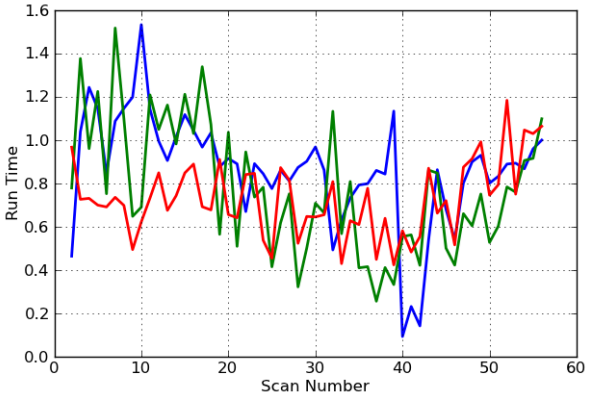
The effort to tune the ICP maximum corresponding point distance parameter (d) produced inconclusive results. The interactions between the parameter and the environmental variables discussed in Section 3.1 proved to be too complex to offer a clear optimal value. The position error in the updated pose estimate, the required runtime and the average number of point pairs was plotted against different values of d for four scans in the path, shown in Figure 3.3. The results indicate that position error and average number of point pairs are slightly positively proportional to d, but runtime is not significantly connected. This is likely because of the number of iterations allowed and the minimization exit criteria. The same three parameters were also plotted for every scan in the path for three different values of d, as shown in Figure 3.2. These plots demonstrate the complexity of optimizing d, as the relationships discussed before are not clear. At times, increasing the number of point pairs improves the results, but not consistently. As the error visualizer shown in Figure 3.4



(a) Position Error for 3 values of d



(b) Average Number of Point Pairs for 3 values of d

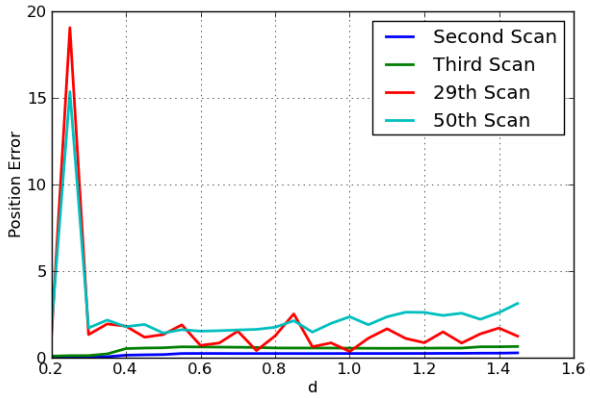


(c) Runtime for 3 values of d

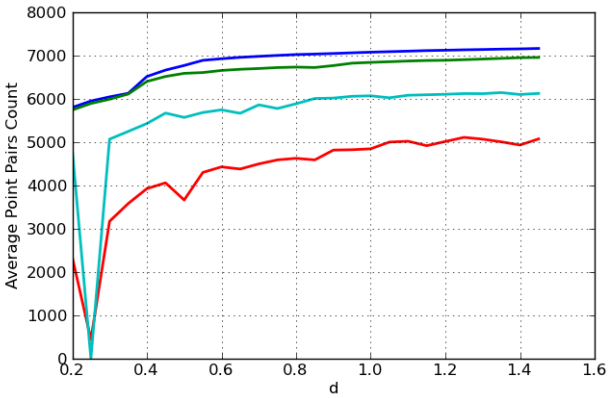
Figure 3.2: Variations in Error, Point Pairs and Runtime with d across All Scans

demonstrates, different values of d perform differently at different points in the scan.

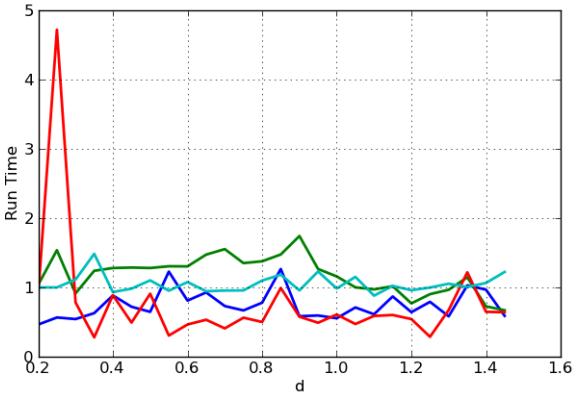
A challenge in this process was deciding how to use the information produced by the



(a) Position Error for 4 scans in path



(b) Average Number of Point Pairs for 4 scans in path



(c) Runtime for 4 scans in path

Figure 3.3: ICP Maximum Corresponding Point Distance Tuning Results

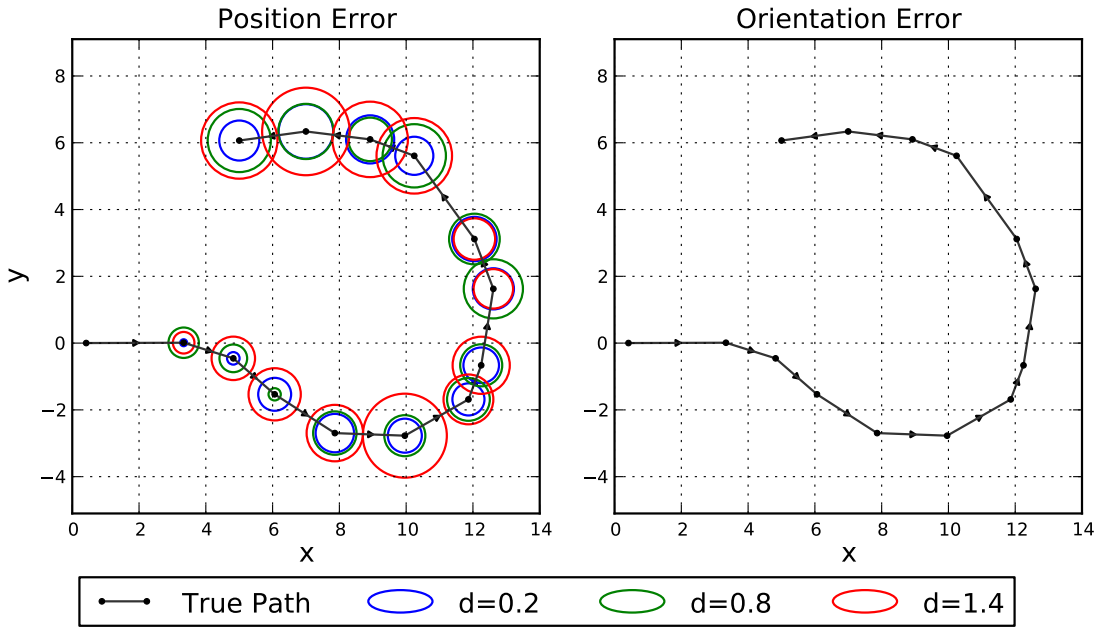


Figure 3.4: Error Visualization of ICP Maximum Corresponding Point Distance Tuning

later scans. The ICP algorithm is sensitive to the initial guess, and the initial guesses for later scans are not only successively worse, but dependent on the quality of the estimations for the previous scans. It proved impossible to meaningfully control all of the compounding variables, test their impacts, and synthesize the results into one optimized parameter within the available time with the testing methods available.

A value of 0.5 m was chosen based on the results. This value is large enough to produce sufficient number of point pairs between scans without including pairs that are not actually corresponding points. However, if the scan frequency is changed from 1Hz as used for these tests, it is likely that the value will have to be adjusted.

3.3 Final Algorithm Evaluation

Computational Performance Analysis

As anticipated, it is possible to run scan matching ICP in under 1 second and process scans as they arrive from the sensor. We can achieve close to real-time performance. The length of time needed to run GraphSLAM depends on two things, the total number of scans, and the

Table 3.2: Time Required to Preprocess Scan, Match Scans (ICP), and Refine Poses (GraphSLAM)

Process	Number of Scans	Required Time (s)
Preprocess Scan (Average)	-	0.005296
Match Scan (Average)	-	0.78651
Refine Poses	5	2.77776
Refine Poses	9	5.15809
Refine Poses	16	7.34996
Refine Poses	35	14.4046
Refine Poses	51	11.3075
Refine Poses	55	10.2169

number of scans since GraphSLAM was last completed. Increasing either of these increases the amount of time required for GraphSLAM to relax the linear graph and improve the estimates. This is expected since the portion of the graph which has already been optimized should require less adjustment in future iterations.

The algorithms were tested on two laptops, one significantly more powerful than the other. Instead of the expected decrease in times on the faster machine, the times were not significantly improved. Checking the CPU usage, the computer was unable to max out its power while running the node, reaching only 50% of it's peak usage. In contrast, the slower laptop was able to maximize it's use of the CPU. At this point, this behaviour is unexplained. One theory is that the version of Ubuntu used is not optimized for the newer CPU found in the powerful laptop.

Error Analysis

The path as estimated by the final algorithm are shown in Figure 3.5. This plot demonstrates that while the refined estimates preserve the shape of the path, the estimates of the z coordinate are not accurate at the end of the data set. Nevertheless, they still represent an improvement over the EKF results which don't change in z. The error data for each estimate of the pose is shown in Table 3.3. The data demonstrates that the euclidean distance between error for the ICP estimate is significantly improved over the EKF estimate. However, running GraphSLAM does not significantly decrease the error when compared to the ICP estimates. The error visualization of the same run is shown in Figure 3.6. Notice that significant error

Table 3.3: Position Errors Present over Path in Successive Pose Estimates

Position

Estimate	Average (m)	Maximum (m)	Minimum (m)
Odometry Poses	1.63	2.48	9.2×10^{-3}
Scan Matching Poses	0.977	2.06	9.0×10^{-3}
Refined Poses	1.17	2.27	9.0×10^{-3}

Orientation

Estimate	Average (m)	Maximum (m)	Minimum (m)
Odometry Poses	2.3×10^{-3}	5.3×10^{-3}	7.6×10^{-4}
Scan Matching Poses	2.6×10^{-2}	6.2×10^{-2}	3.7×10^{-4}
Refined Poses	3.7×10^{-2}	0.117	3.7×10^{-4}

has been redistributed into the early scans in the path, but scans at the end have improved estimates. It is unclear so far whether GraphSLAM can be tuned to improve the estimates enough that it is worth the extra computational resources. We anticipate that it would become more relevant when running on a longer dataset with bigger potential for drift, and with loops in the path.

It is odd that there is so little error present in the orientation estimates. As seen in Table 3.3, the odometry estimates of the orientation of the robot are almost perfect. It is not understood at this time why this is, as noise in being inserted into the data generated by the simulation.

Another problem which we observed is that our method for producing initial estimates for ICP using the difference in the EKF position since the last pose was estimated does not work because of a timing issue in the way that the EKF data is delivered to the slam6drealtime node. While the mathematics of calculating the difference in the EKF estimate is correct, the node does not receive estimates from the EKF that coincide with the scans taken by the laser sensor. To solve this problem, the EKF node needs to be rewritten as a ROS service rather than a broadcaster where slam6drealtime can request the EKF estimate at any time, rather than receive it at specific times. Therefore the results presented here use the EKF output directly to initialize ICP. The prediction is that the other initialization strategy would result in a better result if the timing errors were solved.

The computational framework of an ICP foreground process with a GraphSLAM back-

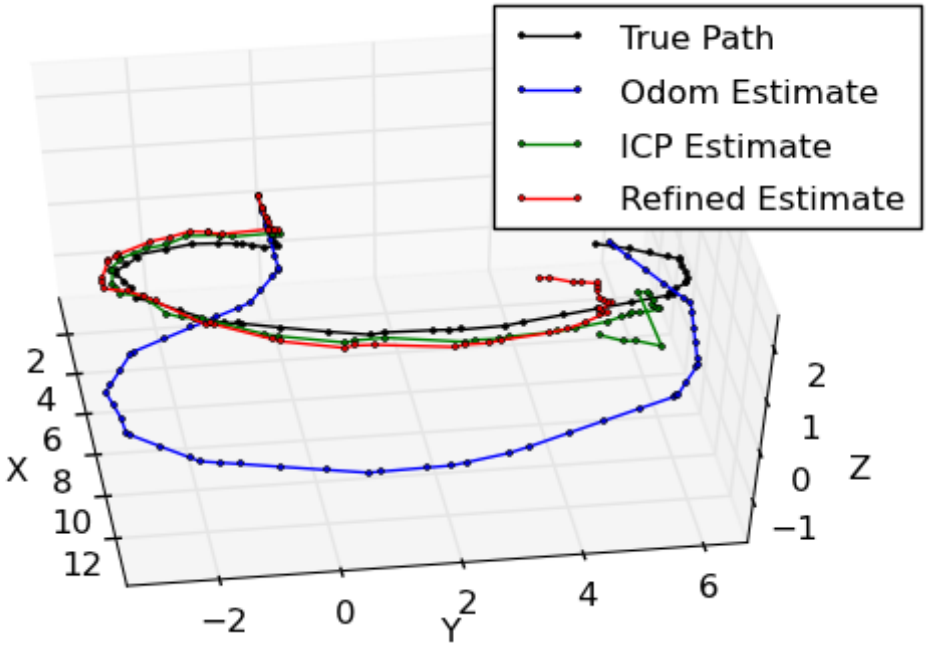


Figure 3.5: Final Algorithm Results in Path Plotting Visualization Tool

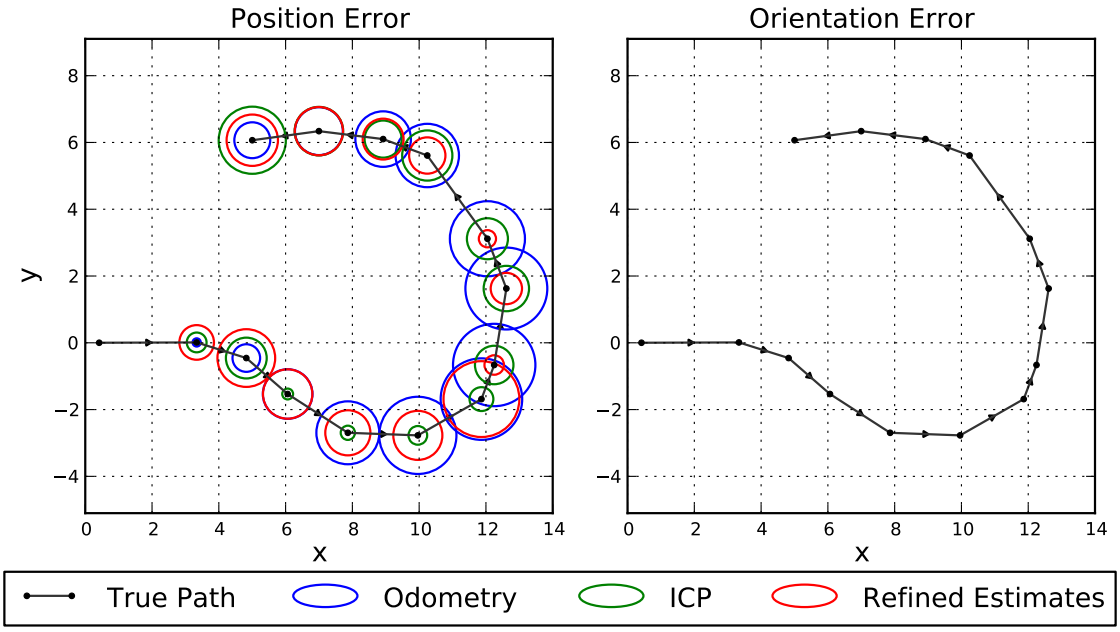


Figure 3.6: Final Algorithm Results in Error Visualization Tool

end process has been proven to be a beneficial and viable structure for a real-time SLAM process with modern computational power. The process results in a significant improvement in pose estimation over odometry and IMU data processed with a EKF filter, and the results are delivered within a reasonable time.

ROS Node Stability

The ROS node runs well on the short data set used for testing (approximately 1 minute with a scan frequency of 1Hz). However, Nuchter's code begins to return errors in both ICP and GraphSLAM process when the node is run on longer data sets, generally after 3-5 minutes. While the node does not crash, the pose estimates delivered can become very wrong. The errors delivered indicate that issues are arising in the inversion or construction of a matrix. The exact cause of these errors has not been determined, as the point at which they begin to occur is not consistent. This numerical instability will be investigated in the coming weeks before the competition.

Functional Requirements Analysis

As a reminder, the functional requirements of the system were as follows.

1. Perform scans of the surrounding area.
2. Perform pre-processing on incoming scan data (to reduce the effects of noise, for example).
3. Determine the pose (location and orientation) of the robot in real time as the robot travels through the environment.
4. Build up a map of the surroundings as the robot travels through the environment.

The first two functional requirements were easily satisfied. Scans are performed using the Velodyne (or a simulation of the Velodyne) and the data is filtered to remove outliers. The outliers that are removed include those that are too close to the robot or too far away. The third requirement was the focus of our efforts in the project as fulfilling the third requirement

lets us construct a map using the amalgamation of the point clouds obtained from the scans. The results shown in the this chapter indicate that the robot's pose can be obtained using the stated design. There is still some work to be done to ensure that these results are robust over longer periods of time and to explain some uncertainties in the collected data. The functional requirements have been successfully met.

4

Recommendations

Before further modifications are done to the algorithms presented in this report the next step should be to collect a set of laser scans in the real world that will reflect similar conditions to those that will be found at the competition. The scans should be correlated with ground truth information from GPS or other measurements. This outdoor data set should be used to further test and refine the algorithms presented. The parameter searches that were run on the testing data should be run again on the outdoor data to confirm the selected parameters. While it is anticipated that the simulation environment will closely mimic the real competition, there will be some modifications that need to be made for the actual sensors.

Once outdoor testing data is available, an effort must be made to resolve the unexplained oddities that were encountered in the results. The mathematical reasons for ICP starting to give unexpected results on long bag files and the orientation estimates that seem too accurate should be examined more closely in the context of real outdoor data.

There are also a number of modifications that could be made to the algorithms within the framework that might achieve better results. In particular there are options that could be considered for each stage in the algorithm framework.

For providing the initial guess for ICP, the Extended Kalman Filter could be easily augmented to include another set of measurements from visual odometry. Visual odometry is a method of using regular cameras to estimate the robot's motion using computer vision techniques. Very briefly, features are extracted from images of the robot's surroundings and the motion of these features from frame to frame are used to estimate the robot's motion.

This would increase the accuracy of the initial guess used to initialize ICP but would require a significant increase in the computational resources required for calculating the initial guess. It's unclear whether the benefit of a better initial guess would be able to counteract not being able to match scans as frequently during the competition.

At the scan-matching stage of the algorithm, another option that could be considered is using the Normal Distributions Transform (NDT) to perform the scan registration in addition to or instead of ICP. NDT is a method of matching normal distributions to concentrations of points in scans and then determining the transforms required to achieve the same distributions in successive scans [10]. Recent work done in the WaveLAB at the University of Waterloo has shown that the use of ICP to initialize NDT can result in a significant decrease in the localization error in 2-dimensions [24]. It is likely that the same results could be achieved in 3-D. It is unclear whether the combination of the two algorithms could be run in real-time so further testing is necessary in this area.

Instead of GraphSLAM, it may be possible to work with some of the known features of the competition grounds to achieve better global optimization for the robot's pose. For instance, it is known that to the left of the starting area for the competition there is a bandstand. If this bandstand could be identified by the robot, then whenever it passes near the feature the global map could be corrected using the bandstand's absolute truth position as a measurement. This strategy requires some very specific optimizations for any given environment but could be very useful for the competition. It could be used in addition to GraphSLAM or the constraints from known features could be incorporated into the linear graph.

5

Project Management

Our project did not go as planned in the Gantt chart submitted in December. Unfortunately, the robot hardware which would be required for outdoor testing was not completed in time for us to perform outdoor testing before this report was handed in. Due to this, our project was completed entirely within the testing simulation environment created in the Fall. Apart from the outdoor testing, we generally followed the plan outlined in December with the addition of more algorithm refinement performed in the simulation evnironment instead of outdoor testing. We progressed slower than expected in completing the implementation of a 6-DOF SLAM algorithm and did not end up trying MRPT SLAM in addition to the 3D-Toolkit.

6

Conclusion

The goal of this project was to develop a system for real-time autonomous robot localization and mapping (SLAM) for use during the NASA Sample Return Robot Challenge. Previous work in the field has been in three separate areas. The first is the filtering and integration of data from wheel odometry and IMU acceleration with filters (like the Kalman Filter used in this project) to obtain estimates of the robot's movement. This technique provides low fidelity estimates at low computational cost. The second involves using scan matching techniques (like ICP) to get more accurate pose estimates at the cost of greater computational effort. The final technique involves using the entire history of scans that the robot has taken to generate an even better set of pose estimates using optimization techniques like GraphSLAM. Once again, this can provide better estimates of position, but is so computationally intense that it has generally only been used in post-processing.

The design team sought to utilize these techniques for their accuracy, but place them in a novel computation structure that provides real-time pose estimates. The final design involves using an Extended Kalman Filter integrating wheel odometry and IMU measurements to initialize scan matching using ICP. After a number of pose estimates have been performed using ICP, GraphSLAM is performed on a background thread to further refine the pose estimates. While GraphSLAM is performed, the robot is still able to localize itself using ICP because of the multi-threaded nature of the node. After GraphSLAM is completed in the background, the adjustments made are applied forward to the pose estimates calculated since it was started.

Good results have been achieved using this technique with some caveats. In general, the technique offers a significant improvement over using just an EKF to localize the robot. Scan matching is performed in real-time (in approximately 1 sec) and provides an improved estimates of the robot's location. GraphSLAM has been shown to provide a small amount of further improvement to the pose estimates but it's unclear whether the increased computational complexity is worth this small improvement. Better pose estimates might be obtained by using the computational power used for GraphSLAM to instead provide better initial estimates to ICP using visual odometry or to run ICP on more frequent scans. There are also some unresolved problems with the results obtained. When running the algorithm on longer datasets, mathematical errors are sometimes encountered while running both ICP and GraphSLAM. These errors will need to be resolved before the algorithms are used on the competition robot.

A combined approach of an EKF, ICP scan matching and GraphSLAM global optimization has been shown to be viable for real-time SLAM for autonomous robots. There are still some issues to be resolved before this approach can be used on the competition robot.

References

[1] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*. ACM Inc., 2001.

[2] NASA, *NASA Centennial Challenge Sample Return Robot Challenge Rules*, 1.1 ed., 2011.

[3] S. Thrun, "Robotic Mapping : A Survey," no. February, 2002.

[4] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "Heuristic-Based Laser Scan Matching for Outdoor 6D SLAM," in *In Advances in artificial intelligence. 28th annual German Conf. on AI*, pp. 304–319, 2005.

[5] A. Nüchter, K. Lingemann, J. Hertzberg, S. Birlinghoven, and D.-S. Augustin, "6D SLAM 3D Mapping Outdoor Environments," *Journal of Field Robotics*, vol. 24, pp. 699–722, 2007.

[6] P. Besl and N. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on pattern analysis and machine intelligence*, pp. 239–256, 1992.

[7] J. Martinez, J. González, J. Morales, A. Mandow, and A. Garcia-Cerezo, "Genetic and ICP laser point matching for 2D mobile robot motion estimation," 2002.

[8] Q. Muhlbauer, K. Kuhnlenz, and M. Buss, "Fusing laser and vision data with a genetic ICP algorithm," in *2008 10th International Conference on Control, Automation, Robotics and Vision*, pp. 1844–1849, Ieee, Dec. 2008.

[9] Y. Jeong, Y. Bok, J. Kim, and I. Kweon, "Complementation of cameras and lasers for accurate 6D SLAM: From correspondences to bundle adjustment," in *Robotics and*

Automation (ICRA), 2011 IEEE International Conference on, pp. 3581–3588, IEEE, 2011.

[10] P. Biber and W. Strasser, "The normal distributions transform: a new approach to laser scan matching," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3, pp. 2743 – 2748 vol.3, oct. 2003.

[11] S. Waslander, "ME 780 - Section 1 - GraphSLAM." 2012.

[12] D. Asmar, *Vision-inertial SLAM using natural features in outdoor environments*. PhD thesis, University of Waterloo, 2007.

[13] K. L. Andreas Nüchter and J. Hertzberg, "Cached k-d tree search for ICP algorithms," 2007.

[14] A. Nüchter, *3D Robotic Mapping - The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*, vol. 52 of *Springer Tracts in Advanced Robotics*. Springer, 2009.

[15] Jacobs University, Osnabruck University, "The 3D Toolkit." `http://slam6d.sourceforge.net/`.

[16] W. B. Sebastian Thrun and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[17] H. Baltzakis, A. Argyros, and P. Trahanias, "Fusion of laser and visual data for robot motion planning and collision avoidance," *Machine Vision and Applications*, vol. 15, pp. 92–100, Dec. 2003.

[18] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," in *Proceedings 2006 IEEE International Conference on Robotics and Automation*, no. 3, pp. 1180–1187, Ieee, 2006.

[19] A. Nüchter, O. Wulf, K. Lingemann, J. Hertzberg, B. Wagner, and H. Surmann, "3D Mapping with Semantic Knowledge," in *IN ROBOCUP INTERNATIONAL SYMPOSIUM*, pp. 335–346, 2006.

[20] Blender WikiBook, "Blender 3D: Noob to Pro/How to Do Procedural Landscape Modeling." http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/How_to_Do_Procedural_Landscape_Modeling.

[21] C. B. Barber and H. Huhdanpaa, "Qhull manual." http://www.qhull.org/html/index.htm#definition.

[22] John Hunter, Darren Dale, Michael Droettboom, "matplotlib." http://matplotlib.sourceforge.net/.

[23] John Hunter, Darren Dale, Michael Droettboom, "mplot3d." http://matplotlib.sourceforge.net/mpl_toolkits/mplot3d/index.html.

[24] A. Das and S. L. Waslander, "Initialization of the Normal Distributive Transform Registration Optimization using the Iterative Closest Point Algorithm," February 2012.