ECE276A Project 2: LiDAR - Based SLAM

Richard Oliveira
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A

I. Introduction

In the context of autonomy, more specifically in robotics, as the scope of many problems in this field increase, it is necessary to accurately track and map the motion of a robot's trajectory and also it's environment, through the use of sensors, such as LiDAR, cameras, wheel encoders and other types of sensors. The problem of estimating the robot's position and environment, is a common problem in robotics and it is known as the *Simultaneous Localization and Mapping* (SLAM) problem[3]. In this project we present a way of approaching the SLAM problem for a *Differential-Drive* robot via the use of measurements coming from wheel encoders, IMU, LiDAR and a RGBD camera (Kinect). In order to facilitate the *Localization* part of SLAM, we make use of the *Iterative Closest Point* (ICP) algorithm [4] to estimate the robot's pose over time while using LiDAR measurements and also use a library called GTSAM [2] which re-formulates the SLAM problem and optimizes the trajectory of our robot. We then build a occupancy grid of the environment, which addresses the *Mapping* part of SLAM, using an optimized trajectory obtained from ICP and LiDAR measurements, and GTSAM. Finally, in order to obtain a more accurate view of the robot's environment, we use the optimized trajectory and also the images captured by the Kinect device to create a texture map of the robot's environment by projecting these camera images onto the occupancy grid which was created earlier.

II. Problem Formulation

Consider a *Differential-Drive* robot, moving in the (x, y) plane, with orientation θ along its z-axis, equipped with the following sensors:

- Wheel Encoders
- IMU
- Hokuyo
- Kinect

The *motion model* of our robot is given by the *Euler Discretization* model [5]:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) = \begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \boldsymbol{x}_t + \tau_t \begin{pmatrix} v_t cos(\theta_t) \\ v_t sin(\theta_t) \\ \omega_t \end{pmatrix} \quad (1)$$

Where (x_t, y_t) denotes the position of the robot at time t and θ_t denotes the orientation of the robot at time t. The linear velocity of the robot at time t is given by v_t and it is obtained from the Wheel Encoders. The angular velocity ω_t, on the other hand, is obtained from IMU measurements. Finally τ_t represents the time between two consecutive time stamps. Our objective is to optimize the trajectory of out robot $(x_{0:K}, y_{0:K}, \theta_{0:K})$ over K time steps. In order to optimally solve this problem, we turn to SLAM, which involves solving the following optimization problem [3]:

$$\min_{\boldsymbol{x}_{1:K}, \boldsymbol{m}} \sum_{t=1}^{K} ||\boldsymbol{z}_t - h(\boldsymbol{x}_t, \boldsymbol{m})||_2^2 + \sum_{t=0}^{K-1} ||\boldsymbol{x}_{t+1} - f(\boldsymbol{x}_t)||_2^2 \quad (2)$$

Given an initial robot state $\boldsymbol{x}_0$, sensor measurements $\boldsymbol{z}_{1:K}$, with observation model $h(\boldsymbol{x}_t, \boldsymbol{m})$ and motion model $f(\boldsymbol{x}_t)$, we wish to estimate the robot's trajectory $\boldsymbol{x}_{1:K}$ and build a map $\boldsymbol{m}$. Our observation model h will change accordingly, depending on the type of measurements $\boldsymbol{z}_{1:K}$ that we are using.

The problem defined in (2) can be solved in different ways. However, there exist a re-formulation of (2), that involves a so called *Factor Graph*. Given a graph object $\mathcal{G} = \{\mathcal{X}, \mathcal{E}\}$, where $\mathcal{X} = \{T_1, \ ... \ , T_K\}$ represents a set of *Nodes* containing the robot's pose T_t at some time t. The set of *Edges* $\mathcal{E} = \{_0T_1, \ ... \ ,_{K-1} T_K\}$, on the other hand represents the *relative* poses between time steps $t+1$ and t. Additionally a pose T_t at some time t is given by:

$$T_t = \begin{pmatrix} R(\theta_t) & \boldsymbol{x}_t \\ \boldsymbol{0}^T & 1 \end{pmatrix} \quad (3)$$

The above formulation yields the following optimization problem:

$$\min_{\{T_i\}} \sum_{(i,j) \in \mathcal{E}} ||W_{ij} e(T_i, T_j)||_2^2 \quad (4)$$

Where W_{ij} is a set of weights between edges i and j in $\mathcal{E}$. The *Factor* $e(T_i, T_j)$ between poses $T_i, T_j \in \mathcal{X}$ is given by:

$$e(T_i, T_j) = log(\bar{T}_{ij}^{-1} T_{ij}^{-1} T_j)^{\vee} \quad (5)$$

Where $\bar{T}_{ij}$ is a set of relative poses obtained by *Odometry*. Given the minimizer for either (2) or (4), which we denote by $T'_{1:K}$, we wish to create an occupancy grid $\boldsymbol{m} \in \mathbb{R}^{N \times N}$, where we model each cell $m_i \in \boldsymbol{m}$ as a Bernoulli random variable. Our goal is to fill each cell m_i in the occupancy grid. The texture map on the other hand, which we denote by $\bar{\boldsymbol{m}} \in \mathbb{R}^{N \times N \times 3}$, consists in finding a transformation A

from a pixel frame $\{P\}$ to the World frame $\{W\}$ such that ${}_{\{W\}}A_{\{P\}}x_{\{P\}} = y_{\{W\}}$. We then seek to fill each cell of the texture map $\bar{\boldsymbol{m}}_i$ in order to obtain a clearer view of the robot's environment.

III. Technical Approach

We approach the problem in the following manner:

- *Data Pre-Processing*
- *Encoder and IMU Odometry*
- *Point-cloud registration via ICP*
  - Warm-up
  - Scan Matching
- *Occupancy and Texture mapping*
- *Pose graph optimization and loop closure*
  - Factor Graph in GTSAM
  - Loop closure detection

A. Data Pre-Processing

In this section we discuss how we process the data which is collected by the sensors mounted on the robot.
The Wheel Encoder counts the rotation of the four wheels at 40 Hz. We have that one rotation of the wheel corresponds to l meters of distance travelled. We have 360 tics per revolution of the wheel, and it travels at 0.0022 meters per tic. The Encoder counts are represented in this manner $\{FR,\ FL,\ RR,\ RL\}$ (FR means "Forward Right", FL means "Forward Left", RR means "Rear Right", RL means "Rear Left"). We can establish that the Right wheel travels a distance of $\frac{(FR+RR)}{2} * 0.0022m$, and the Left wheel travels a distance of $\frac{(FL+RL)}{2} * 0.0022m$. Given the equation:

$$v_t \approx \frac{\pi dz}{n\tau_t} \tag{6}$$

Where d corresponds to the diameter of the wheels, n is the tics per revolution, and z is an Encoder count, which is given by data. Therefore, we reach the conclusion that the linear velocity at which the robot travels is given by $v_t \approx \frac{0.0022m}{\tau_t}$. Additionally we choose the linear velocity v_t to be the average between the linear velocity of the two right wheels and the linear velocity of the two left wheels. The IMU data on the other hand requires minimal pre-processing since we are only interested in the angular velocity ω_t around the robot's z-axis.

The Hokuyo device gives us LiDAR measurements. It is equipped with a 270° degree field of view. A LiDAR scan at some time t is in the form $(r_1^t, \phi_1^t),\ \ldots\ ,\ (r_{1081}^t, \phi_{1081}^t)$. Here, each i^{th} measurement is converted into *rectangular* coordinates by using the conversion:

$$\begin{aligned} x_t &= r_i^t cos(\phi_i^t) \\ y_t &= r_i^t sin(\phi_i^t) \\ z_t &= 0.51435m \end{aligned}$$

Here the coordinate z_t was chosen to be the height of the Hokuyo device.

B. Encoder and IMU Odometry

Before we begin talking about this section, we mention that given the time stamps of the Wheel encoder $\boldsymbol{t}_E \in \mathbb{R}^N$ and the time stamps of the IMU data $\boldsymbol{t}_{IMU} \in \mathbb{R}^D$. Given that $N < D$, we wish to sync the data, such that the IMU measurements that are taken will be as close as to when the Wheel Encoder recorded wheel counts. We do this by iterating through the time stamps of the Encoder $\boldsymbol{t}_E \in \mathbb{R}^N$, and we compute the difference between the first time stamp of $\boldsymbol{t}_E$ and all the elements in $\boldsymbol{t}_{IMU}$, we then compute the index j^* of $\boldsymbol{t}_{IMU}$ which gives us the smallest difference in order to see which time stamp in $\boldsymbol{t}_{IMU}$ is closest to $\boldsymbol{t}_E$. After, we construct a new set of IMU measurements indexed at j^*. We repeat this process for all time stamps in $\boldsymbol{t}_E$.

In this portion of the project, we simply pre-process the data as discussed in the *Data Pre-Processing* section, and we use the *motion model* specified by (1) to compute the trajectory of the robot over T time stamps (also known as *Dead Reckoning*).

C. Point-cloud registration via ICP

This section is divided into two parts: Warm-up and Scan Matching. The Warm-up section illustrates how the ICP algorithm was implemented. The Scan matching section, discusses how a better estimate of the robot's trajectory over T time stamps was obtained via the ICP algorithm, combined with LiDAR measurements.

1) Warm-up: The ICP algorithm is divided into two steps: *Data Association* step and *Kabsch Algorithm* step.

The general *Point Cloud registration problem* is the following: given two point clouds $\{\boldsymbol{m}_i\} \in \mathbb{R}^d$ (target) and $\{\boldsymbol{z}_j\} \in \mathbb{R}^d$ (source), find the transformation $\boldsymbol{p} \in \mathbb{R}^d$, $R \in SO(d)$ and data association Δ such that the following cost is minimized:

$$\min_{\boldsymbol{p}\in\mathbb{R}^d, R\in SO(d), \Delta} \sum_{(i,j)\in\Delta} w_{ij} ||(R\boldsymbol{z}_j + \boldsymbol{p}) - \boldsymbol{m}_i||_2^2 \tag{7}$$

$$SO(d) = \{R \in \mathbb{R}^{d\times d} \mid R^T R = I,\ det(R) = 1\}$$

In the context of our problem, we define a source, which is also a point cloud $\boldsymbol{z}_j$ of points in $\mathbb{R}^3$ and a target $\boldsymbol{m}_i$ of points in $\mathbb{R}^3$. As mentioned earlier we need to perform the *Data Association* step, where we associate each point in the source with a point in the taregt, such that the two point clouds have the same number of points. Mathematically, this step involves solving the following optimization problem at some iteration k:

$$i \leftrightarrow arg\min_j ||\boldsymbol{m}_i - (R_k \boldsymbol{z}_j + \boldsymbol{p}_k)||_2^2 \tag{8}$$

Where $\boldsymbol{p}_k \in \mathbb{R}^3$ and $R_k \in SO(3)$. Once (8) is solved, we move on to the *Kabsch Algorithm* step. The *Kabsch Algorithm* assumes that both point clouds, target $\boldsymbol{m}_i$ and source $\boldsymbol{z}_i$ are associated. Therefore (7) reduces to:

$$\min_{\boldsymbol{p}\in\mathbb{R}^d, R\in SO(d)} f(R,\boldsymbol{p}) := \sum_i w_i ||(R\boldsymbol{z}_i + \boldsymbol{p}) - \boldsymbol{m}_i||_2^2 \quad (9)$$

We start by defining the centroids of the two point clouds:

$$\bar{\boldsymbol{m}} = \frac{\sum_i w_i \boldsymbol{m}_i}{\sum_i w_i}$$
$$\bar{\boldsymbol{z}} = \frac{\sum_i w_i \boldsymbol{z}_i}{\sum_i w_i} \quad (10)$$

We set $\nabla_{\boldsymbol{p}} f(R,\boldsymbol{p}) = 0$, and solve for $\boldsymbol{p}$. We obtain:

$$\boldsymbol{p}^* = \bar{\boldsymbol{m}} - R\bar{\boldsymbol{z}} \quad (11)$$

By replacing (10) into (9), leads to *Wahba's Problem*:

$$\max_{R\in SO(3)} tr(Q^T R) \quad (12)$$

Where $Q = \sum_i w_i \delta\boldsymbol{m}_i \delta\boldsymbol{z}_i^T$. Here $\delta\boldsymbol{m}_i$ and $\delta\boldsymbol{z}_i$ are the points $\boldsymbol{m}_i$ and $\boldsymbol{z}_i$ shifted to their respective centroids:

$$\delta\boldsymbol{m}_i = \boldsymbol{m}_i - \bar{\boldsymbol{m}}$$
$$\delta\boldsymbol{z}_i = \boldsymbol{z}_i - \bar{\boldsymbol{z}}$$

(12) is solved by setting the weights $w_i = 1$. Consider the *Singular Value decomposition* of $Q = U\Sigma V^T$. The optimal matrix $R^* \in SO(3)$ ($d = 3$) that maximizes (12) is given by:

$$R^* = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{pmatrix} V^T \quad (13)$$

Once, R^* is computed, we can use (11) to compute $\boldsymbol{p}^*$.
We are now ready to state the ICP algorithm. Given two point clouds $\boldsymbol{m}_i \in \mathbb{R}^3$ (target) and $\boldsymbol{z}_j \in \mathbb{R}^3$ (source), where $i = 1, ..., N$ and $j = 1, ..., M$ ($M > N$) and initial estimates at iteration $k = 0$, $R_0 = I \in \mathbb{R}^{3\times3}$ and $\boldsymbol{p}_0 = \bar{\boldsymbol{m}} - \bar{\boldsymbol{z}} \in \mathbb{R}^3$. We iterate between the following steps (Step 1 - Step 5) for K iterations:

$$i \leftrightarrow arg\min_j ||\boldsymbol{m}_i - (R_k\boldsymbol{z}_j + \boldsymbol{p}_k)||_2^2 \longrightarrow (\boldsymbol{z}_i, \tilde{\boldsymbol{m}}_i) \quad \text{(Step 1)}$$

$$\bar{\boldsymbol{m}} = \frac{\sum_i \tilde{\boldsymbol{m}}_i}{N},\ \bar{\boldsymbol{z}} = \frac{\sum_i \boldsymbol{z}_i}{N} \quad \text{(Step 2)}$$

$$\delta\tilde{\boldsymbol{m}}_i = \tilde{\boldsymbol{m}}_i - \bar{\boldsymbol{m}},\ \delta\boldsymbol{z}_i = \boldsymbol{z}_i - \bar{\boldsymbol{z}} \quad \text{(Step 3)}$$

$$Q = \sum_i \delta\boldsymbol{m}_i \delta\boldsymbol{z}_i^T,\ Q = U\Sigma V^T \quad \text{(Step 4)}$$

$$R_{k+1} = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{pmatrix} V^T,\ \boldsymbol{p}_{k+1} = \bar{\boldsymbol{m}} - R_{k+1}\bar{\boldsymbol{z}} \quad \text{(Step 5)}$$

After K iterations we obtain R_K and $\boldsymbol{p}_K$, which are then used to shift the source point clouds given in the data sets for "*drill*" and "*liq container*" to their respective target.

2) Scan Matching: Once the ICP algorithm has been implemented, we can discuss the Scan Matching process. Here we combine Odometry data (Wheel Encoders and IMU) with LiDAR scans at each time step t. Previously we have mentioned in the *Encoder and IMU Odometry* section, the concept of data synchronization. The same will be done in Scan Matching, where we try to match each time step in the LiDAR time stamps to the closest time step in the Wheel Encoder time stamps. We pre-process the data for the LiDAR measurements accordingly based on the procedure that is explained in the *Data Pre-Processing* section.
First, we introduce the *observation model* for the Hokuyo device [5]:

$$\bar{\boldsymbol{m}} = h((\boldsymbol{x}, R), \boldsymbol{p}) = R^T(\boldsymbol{m} - \boldsymbol{p}) \quad (14)$$

Where $\bar{\boldsymbol{m}} \in \mathbb{R}^3$ represents an observation in LiDAR frame $\{L\}$, $\boldsymbol{m} \in \mathbb{R}^3$ represents the point observed in World frame coordinates $\{W\}$, $\boldsymbol{p} \in \mathbb{R}^3$ represents the position of the device with respect to the body frame of the robot $\{B\}$, and R is simply the rotation matrix of the device. The Scan Matching process begins by considering two individual scans, one taken at some time step t and another one taken at a time step $t + 1$. The two scans will give us a set of measurements in *spherical* coordinates, which will be then converted into *rectangular* coordinates $\{(x_i^t, y_i^t, z_i^t)\}$ and $\{(x_j^{t+1}, y_j^{t+1}, z_j^t)\}$. The two scans represent a set of point clouds $\boldsymbol{z}_i$ at time t and $\boldsymbol{m}_j$ at time $t + 1$. Our goal is to compute the relative pose ${}_tT_{t+1}$ between the source and the target, such that the pose of the robot $T_{t+1}^{'}$ at the next time step $t+1$ can be computed as follows:

$$T_{t+1}^{'} \approx T_t^{'}\,{}_tT_{t+1} \quad (15)$$

Computing the relative pose ${}_tT_{t+1}$ involves the use of the ICP Algorithm. First, we convert our scan matches in *rectangular coordinates*, then to body frame coordinates of the robot $\{B\}$. Rearranging (14) we get points $\boldsymbol{z}_i^L$, $\boldsymbol{m}_j^L$ in LiDAR coordinates $\{L\}$, then we shift these points by an amount $0.14965m$ along the y-direction (since we have defined the center of our robot to be it's *geometric* center) to convert the measurements from coordinates $\{L\}$ to body frame coordinates $\{B\}$. Thus leaving us with $\boldsymbol{z}_i^B$, $\boldsymbol{m}_j^B$. Using $\boldsymbol{z}_i^B$ as a target, $\boldsymbol{m}_j^B$ as a source, and initializing $R_0 = R_t^{-1}R_{t+1}$ and $\boldsymbol{p}_0 = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ we run the ICP algorithm (iterating between Steps 1 - 5) for $K = 50$ iterations. We specify that $\boldsymbol{x}_t$ is the position of the robot at time t estimated by *Odometry* and R_t is the rotation matrix of the robot at time t also estimated by *Odometry*. The ICP Algorithm returns a relative pose ${}_tT_{t+1}$ in frame $\{B\}$. Finally we use equation (15) to obtain a "optimal" pose of the robot at time $t + 1$. We repeat this process for T time steps and we obtain the optimized trajectory $T_{0:T}^{'}$ which can be used for the occupancy grid and texture mapping.

D. Occupancy and Texture mapping

As stated in the *Problem Formulation* we begin our occupancy grid procedure by considering a grid $M \in \mathbb{R}^{N\times N}$,

where each cell m_i can be modeled as a Bernoulli Random Variable. This probabilistic approach to the occupancy grid problem is necessary such that we can update each cell as being free or occupied in a less naive way. Additionally we assume that each cell m_i is conditionally statistically independent of other cells m_j given the robot's trajectory $\boldsymbol{x}_{1:T}$. We can therefore track the probability distribution $p(m_i|\boldsymbol{z}_{1:T}, \boldsymbol{x}_{0:T})$ for each cell separately. This formulation yields the following *log odds* update rule:

$$\frac{p((m_i = 1|\boldsymbol{z}_t, \boldsymbol{x}_t))}{p((m_i = -1|\boldsymbol{z}_t, \boldsymbol{x}_t))} = \begin{cases} +log(4) & , \boldsymbol{z}_t \rightarrow \ m_i \text{ IS occupied} \\ -log(4) & , \boldsymbol{z}_t \rightarrow \ m_i \text{ NOT occupied} \end{cases} \tag{16}$$

Here, we add $+log(4)$ if $\boldsymbol{z}_t$ indicates that cell m_i is occupied, and $-log(4)$ if $\boldsymbol{z}_t$ indicates that cell m_i is free.
Given LiDAR scans in *rectangular* coordinates at some time t, in frame $\{L\}$, $\{(x_i^t, y_i^t, z_i^t)\}$. We use *bresenham2D* to obtain the coordinates of cells m_i for which the ray coming from the LiDAR passes through. We need to specify the starting position of the ray in coordinates $\{L\}$, and the end position of the ray in coordinates $\{B\}$. The coordinates obtained in frame $\{L\}$ are converted to frame $\{B\}$ by using the estimated poses from ICP, $T_{1:T}^{'}$. Once we have both, initial and final position of the ray in frame $\{B\}$, we scale them by an amount $1/res$ (where res is the resolution of our grid). We then use *bresenham2D* to obtain the coordinates of cells for which the ray passes through. Finally we follow the *log odds* update rule for occupied and free cells. We repeat this process for all T measurements that the LiDAR provides.

In the context of texture mapping, we consider the data given by the Kinect device. This RGBD camera provides RGBD images of size 480x640 and also Disparity images. The camera is located at $(0.18, 0.005, 0.36)m$ with respect to the robot center and it has an orientation of 0 rad roll, 0.36 rad pitch and 0.021 rad yaw. We are also provided with time stamps for both RGBD images and Disparity images. Just like how we did in the *Encoder and IMU Odometry* and *Scan Matching* section, we will sync the Disparity images time stamps, and estimated poses with the RGBD time stamps. Additionally the intrinsic parameters of the camera are given by the following matrix:

$$K = \begin{pmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{pmatrix} \tag{17}$$

Since the depth camera and the RGBD camera are not in the same location. Given values d at pixel location (i, j) of the Disparity image, we use the following conversions to obtain the depth and pixel location $(rgbi, rgbj)$ of the associated RGBD color:

$$\begin{aligned} dd &= (0.00304d + 3.31) \\ depth &= \frac{1.03}{dd} \\ rgbi &= (526.37i + 192767877.07dd)/585.051 \\ rgbj &= (526.37j + 16662)/585.051 \end{aligned} \tag{17}$$

As stated in the *Problem Formulation* section, we are interested in projecting these images onto the World frame $\{W\}$. Starting from the conversion in (17) we transform the $(rgbi, rgbj)$ pixel locations in the *optical* frame according to the following transformation:

$$\begin{pmatrix} X_o \\ Y_o \\ Z_o \end{pmatrix} = K^{-1} \begin{pmatrix} rgbi \\ rgbj \\ 1 \end{pmatrix} * depth \tag{17}$$

Then we use the following transformation which converts from *optical* frame coordinates to *regular* camera frame coordinates:

$$\begin{pmatrix} X_r \\ Y_r \\ Z_r \end{pmatrix} =_o R_r^{-1} \begin{pmatrix} X_o \\ Y_o \\ Z_o \end{pmatrix} \tag{18}$$

From *regular* frame coordinates we go to the robot - body frame coordinates $\{B\}$ by using the known position of the camera with respect to the origin of the body frame. Given the roll, pitch and yaw angles in radians we construct the rotation matrices R_y and R_z for the pitch and yaw angles respectively, thus yielding the final transformation which takes us from the $\{B\}$ frame to the world frame $\{W\}$:

$$\begin{pmatrix} X_W \\ Y_W \\ Z_W \end{pmatrix} = R_y R_z \begin{pmatrix} X_B \\ Y_B \\ Z_B \end{pmatrix} \tag{19}$$

Now that we have the coordinates in frame $\{W\}$, we project these values into an occupancy grid $\bar{\boldsymbol{m}} \in \mathbb{R}^{N \times N \times 3}$. We note that values outside the range of the occupancy grid size are only possible once the coordinates have been transformed to $\{W\}$, therefore clipping is employed. Since the camera is located at a certain height with respect to the frame $\{B\}$, the converted $\{W\}$ frame pixel coordinates will have an elevation value associated to them. Since we only aim to project the map onto the floor, we threshold all indices with z values greater than the height of the camera with respect to the robot's body frame origin.

E. Pose graph optimization and loop closure

We begin the *Pose graph optimization and loop closure* section by addressing (4). Given a set of optimized poses $T_{1:T}^{'}$ obtained from either *Odometry* or *Scan Matching*, we aim to compute another set of optimized poses $T_{1:T}^{''}$ by using the library GTSAM, which solves problem (4).

1) Factor Graph in GTSAM: In this section we simply construct a *Pose* graph with $T^{'}_{1:T}$ obtained from *Scan Matching*. Using GTSAM we build a graph object $\mathcal{G}^{'} = \{\mathcal{X}^{'}, \mathcal{E}^{'}\}$, where $\mathcal{X}^{'} = \{T^{'}_1, \ \dots \ , T'_K\}$ represents a set of *Nodes* containing the robot's pose $T^{'}_t$ at some time t. The set of *Edges* $\mathcal{E}^{'} = \{{}_0T^{'}_1, \ \dots \ , {}_{K-1}T^{'}_K\}$, on the other hand represents the *relative* poses between time steps $t+1$ and t.

2) Loop closure detection: In this section we explore loop closure methods using GTSAM. We seek to add edges in the set $\mathcal{E}^{'}$ via *Fixed-interval loop closure*. This means that for every 10 poses, at some time $t = 1$, for example, we create an *edge* ${}_1T^{'}_{11}$ between poses $T^{'}_1$ and $T^{'}_{11}$. In addition, all poses in between $T^{'}_1$ and $T^{'}_{11}$ are then connected to pose $T^{'}_{11}$, and we add the following poses to the set $\mathcal{E}^{'}$: ${}_2T^{'}_{11}$, ${}_3T^{'}_{11}$, ..., ${}_{10}T^{'}_{11}$.

Once this scheme is complete for all poses in $\mathcal{X}^{'}$ we solve (4) by defining $e(T_i, T_j)$ to be equal to (5). The GTSAM optimizer provides us with several techniques for solving (4). In this project we choose to use the *Gauss-Newton* method.

Since (5) can be written in the form:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{e}^T(\boldsymbol{x})\boldsymbol{e}(\boldsymbol{x}) \tag{20}$$

The *Gauss-Newton* update at some iteration k, with varying step size α_k is given by:

$$\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \alpha_k \delta \boldsymbol{x} \tag{21}$$

Here the descend direction $\delta \boldsymbol{x}$ is given by the following relation:

$$(\frac{\partial \boldsymbol{e}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}_k})^T(\frac{\partial \boldsymbol{e}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}_k})\delta \boldsymbol{x} = -(\frac{\partial \boldsymbol{e}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}_k})^T\boldsymbol{e}(\boldsymbol{x}_k) \tag{22}$$

We solve (4) and we obtain an optimized sequence of poses $T^{''}_{1:T}$. This sequence will then be used to re-construct a new occupancy grid and texture map.

IV. Results

For the results section we first introduce the results for *Encoder and IMU Odometry*, followed by the results in the Warm-up part of *Point-cloud registration via ICP*. Lastly we present our results for the occupancy grid and texture mapping for the poses obtained from *Scan Matching* and from *Pose graph optimization and loop closure*.

A. Encoder and IMU Odometry

After computing the *Dead Reckoning Trajectory* trajectory over T time stamps, Figure-1 illustrates the motion of the robot for both Data Set 20 and Data Set 21 in the (x, y) plane.

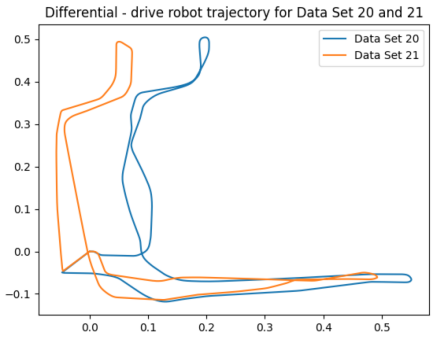

Fig. 1. *Dead Reckoning* for Data Set 20 and Data Set 21

Here we are simply computing the *motion model* defined in (1) at each time step $t+1$.

B. Point-cloud registration via ICP

1) Warm Up: After $K \approx 100$ iterations we obtain a rotation R_K and a translation $\boldsymbol{p}_K$. These are used to shift the source (*canonical object model*) point clouds given in the data sets for "drill" and "liq container" to their respective target. Figure-2 to Figure-9 illustrates the final point cloud alignment.

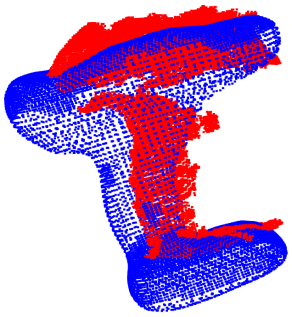

Fig. 2. Drill point cloud 0

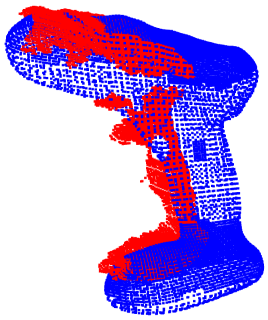

Fig. 3. Drill point cloud 1

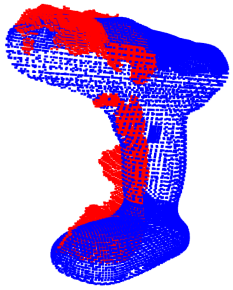

Fig. 4. Drill point cloud 2

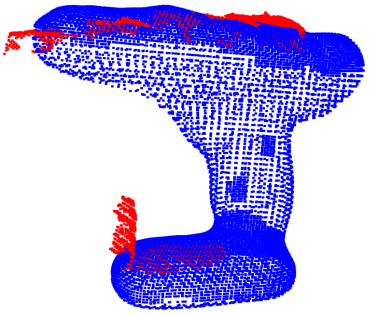

Fig. 5. Drill point cloud 3

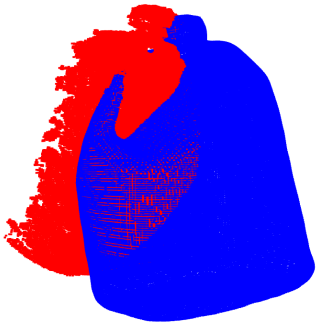

Fig. 6. Liquid Container point cloud 0

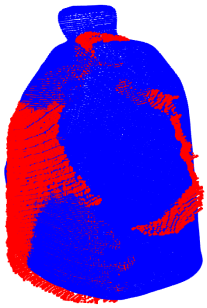

Fig. 7. Liquid Container point cloud 1

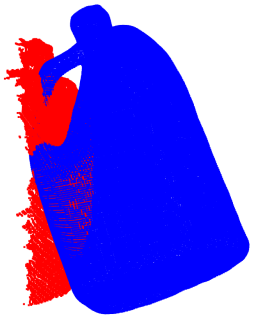

Fig. 8. Liquid Container point cloud 2

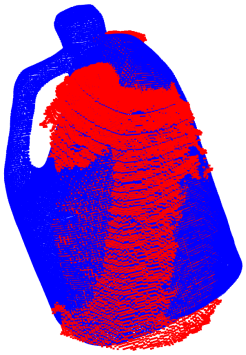

Fig. 9. Liquid Container point cloud 3

We observe that the ICP Algorithm aligns the point clouds fairly well. After several re-iterations of the ICP Algorithm, initializing $R_0 = I$ and $\boldsymbol{p}_0 = \bar{\boldsymbol{m}} - \bar{\boldsymbol{z}}$ seemed to work best rather than choosing $\boldsymbol{p}_0 = [0,\ 0,\ 0]^T$ or choosing $\boldsymbol{p}_0 = [0,\ 0,\ \frac{1}{2}]^T$. We also note that a early stopping point criterion was employed. At iteration k we compute an average loss $\mathcal{L}_k = \frac{1}{N}||(R_k \boldsymbol{z}_k + \boldsymbol{p}_k) - \boldsymbol{m}_k||_2$ and define a threshold $\delta = 10^{-6}$. If the following bound is true we stop and exit the ICP Algorithm early at iteration $k+1$:

$$\mathcal{L}_k - \mathcal{L}_{k+1} < \delta$$

Here the point clouds $\boldsymbol{z}_k \in \mathbb{R}^{3\times N}$ and $\boldsymbol{m}_k \in \mathbb{R}^{3\times N}$ are already associated.

C. Scan Matching

Here we illustrate the occupancy grid results and the texture mapping results after obtaining an optimized sequence of poses from the *Scan Matching* section of this project. Figure 10 and Figure 11 demonstrate the occupancy grid for the *first* LiDAR scan for Dataset 20 and Dataset 21 respectively. Figure 12 and Figure 13, shows the full occupancy grid for all LiDAR scans over T time steps for Dataset 20 and Dataset 21 respectively.

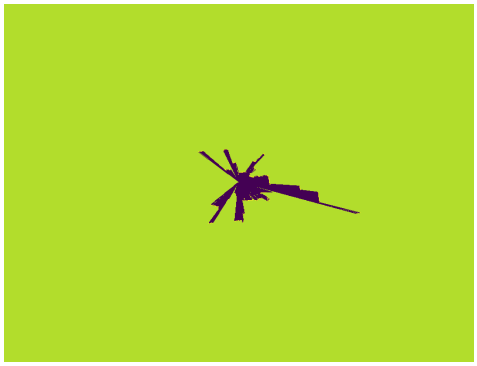

Fig. 10. Occupancy Grid of first LiDAR measurement Dataset 20

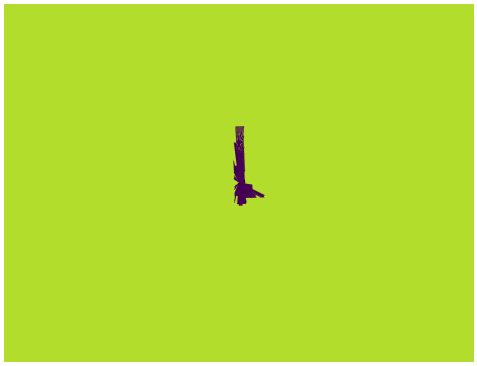

Fig. 11. Occupancy Grid of first LiDAR measurement Dataset 21

Fig. 12. Full LiDAR Occupancy Grid measurement Dataset 20

Fig. 13. Full LiDAR Occupancy Grid measurement Dataset 21

Texture mapping results for both Dataset 20 and Dataset 21 are shown in Figure 14 and Figure 15 respectively.

Fig. 14. Scan Matching Texture map for Dataset 20

Fig. 15. Scan Matching Texture map for Dataset 21

When observing the full occupancy grids, we notice that it's shape and overall form closely resembles the trajectory taken from Scan Matching. There are instances where, some of the rays come out of the building, and this might be due to some small bug in the implementation, since the rays coming from the LiDAR device should not escape an enclosed space. The texture maps are almost formed, only one portion of it seems visible.

D. Pose graph optimization and loop closure

Here we present the occupancy grid and texture mapping after performing *Pose graph optimization and loop closure*. Given an optimized pose sequence over T time steps obtained from GTSAM, we create an occupancy grid and texture map over this sequence. Figure 16 and Figure 17 illustrates the full occupancy grid for Data Set 20 and Data Set 21 respectively. Figure 18 and Figure 19 illustrates the full texture map for Data Set 20 and Data Set 21 respectively.

Fig. 16. GTSAM Occupancy Grid for Dataset 20

Fig. 17. GTSAM Occupancy Grid for Dataset 21

Fig. 18. GTSAM Texture map for Dataset 20

Fig. 19. GTSAM Texture map for Dataset 21

By observing these plots, we notice that for the GTSAM texture map, for Data Set 20 specifically, the resolution that was chosen happened to be too small thus creating a very small texture map. When performing *Pose graph optimization and loop closure* we notice that the occupancy grids and texture maps do not change too much from *Scan Matching* results. Meaning that either the *Scan Matching* process was effective enough in producing already "optimal" trajectories, or there is an error in formulating the *Loop closure* scheme.

REFERENCES

[1] "https://github.com/borglab/gtsam"
[2] "https://gtsam.org/tutorials/intro.htmlmagicparlabel-65412"
[3] slide14 "https://natanaso.github.io/ece276a/ref/ECE276A/5/FactorGraphSLAM.pdf" slide 4
[4] slide25 "https://natanaso.github.io/ece276a/ref/ECE276A/6/LocalizationOdometry.pdf" slide 12
[5] "https://natanaso.github.io/ece276a/ref/ECE276A/4/MotionAndObservationModels.pdf" slide 13