Quantitative Engineering Analysis Final Project: NEATO SLAM

Naomi Chiu and Jonathan Zerez

December 2018

1 Introduction and Contextualization

Our project is based off of an autonomous racecar created by AMZ, the Formula SAE project team from ETH Zurich, as seen in this video. This application was extremely interesting to us as given that we are both part of Olin's Formula SAE team, and given the increasing relevancy of autonomous vehicles within the automotive industry. For our project, we wished to learn more about the analysis that went into developing the autonomy of recognizing an unknown course and self-traversal, also known as SLAM or Simultaneous Localization And Mapping. We hoped to achieve a model of the autonomous racecar using a NEATO that would be able to traverse an unknown course autonomously and use the map that it created from its initial pass for subsequent laps.

In our project, we used a NEATO vacuum cleaner robot, which comes prefitted with a LIDAR sensor, as well as wheel encoders to help with the localization of the robot in a global coordinate frame. The robot is placed in an unknown location, and must navigate and make a map of a "race track" defined by a series of cones. Here is a short video demonstrating a testing setup, as well as the output of the control software that was written.

2 Mathematical Background

This section will outline the concepts behind the mathematical tools used in our project.

2.1 Transformation Matrices and Reference Frames

Transformation matrices are an integral part of dealing with data in multiple reference frames. This project involves measuring points in a moving local reference frame and expressing/storing them in a global, inertial reference frame. As such, transformation matrices are an integral part of this project and is the basis upon which everything else is built.

When multiplying a vector or a set of vectors by a matrix, it can be thought of as changing the space that said vectors exist in. For a given matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} i_x, j_x \\ i_y, j_y \end{bmatrix}$$

Multiplying any 2D column vector $\mathbf{v}$ with the matrix $\mathbf{A}$, ends up transforming $\hat{\mathbf{i}}$ component to land on the coordinate $\begin{bmatrix} i_x \\ i_y \end{bmatrix}$ and $\hat{\mathbf{j}}$ component to land on the coordinate $\begin{bmatrix} j_x \\ j_y \end{bmatrix}$. By changing the building blocks of how a vector is expressed, it is possible to transform where the vector lies. Rotating a 2D column vector by θ degrees in the counter-clockwise can be accomplished by the following matrix:

$$\mathbf{R} = \begin{bmatrix} \cos\theta, -\sin\theta \\ \sin\theta, \cos\theta \end{bmatrix}$$

The translation of a vector can be expressed as a matrix product with the introduction of a third dimension. For a matrix $\mathbf{w}$ with points x in the $\hat{\mathbf{i}}$ direction and y in the $\hat{\mathbf{j}}$, a third $\hat{\mathbf{k}}$ dimension needs to be added with a value of 1. When multiplied by a translation matrix $\mathbf{B}$, the third dimensional component will be scaled such that it has the appropriate $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ components, so as to "push" the original vector $\mathbf{w}$.

$$\mathbf{w} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 1, 0, x' \\ 0, 1, y' \\ 0, 0, 1 \end{bmatrix}$$

$$\mathbf{Bw} = \begin{bmatrix} x + x' \\ y + y' \\ 1 \end{bmatrix}$$

The translation of a vector can also be expressed as a matrix sum. This process is a lot simpler than the one previously examined, as it does not require the addition of a third dimension.

$$\mathbf{w} = \begin{bmatrix} x \\ y \end{bmatrix} \mathbf{B} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\mathbf{w} + \mathbf{B} = \begin{bmatrix} x + x' \\ y + y' \end{bmatrix}$$

2.2 Singular Value Decomposition

Singular Value Decomposition (SVD) is an important tool that can be used in order to find important qualities of any matrix. In this project, it is a tool that is used in the Iterative Closest Point method, which is detailed in Section 2.4. SVD expresses a matrix $\mathbf{S}$ as the product of three special matrices, such that:

$$\mathbf{S} = \mathbf{U}\Sigma\mathbf{V}^T$$

$\mathbf{S}$ can be any matrix, so it very well can both stretch/scale, as well as rotate any input matrix given to operate on. Singular Value Decomposition decomposes a given matrix $\mathbf{S}$ by expressing the entire transformation as a three step process: a rotation, a scaling, and then finally another rotation. This means that $\mathbf{U}$ and $\mathbf{V}$ are rotation matrices, and thus by definition, orthogonal matrices. This also means that Σ is a diagonal matrix, whose entries are called "Singular Values".

Given that the transpose of an orthogonal matrix is equal to its inverse, and the definition of eigenvectors

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

It is simple to prove that the singular values are the square roots of the eigenvalues of the matrix $\mathbf{S}^T\mathbf{S}$.

$$\begin{aligned}
&(1) & \mathbf{S} &= \mathbf{U}\Sigma\mathbf{V}^T \\
&(2) & \mathbf{S}^T\mathbf{S} &= (\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T) \\
&(3) & &= \mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T \\
&(4) & &= \mathbf{V}\Sigma^T\Sigma\mathbf{V}^T \\
&(5) & \mathbf{S^T S V} &= \mathbf{V}\Sigma^T
\end{aligned}$$

It is important to note that the simplification made in (3) can be made because by definition as $\mathbf{U}$ is an orthogonal matrix, which means that

$$\mathbf{U}^T = \mathbf{U}^{-1}$$

It is also important to note that because Σ is a diagonal matrix, the expression $\Sigma^T\Sigma$ is equivalent to squaring the entire matrix element-wise, thus squaring the Singular values held within the diagonal of Σ. The main conclusion that needs to be drawn from all of this is that $\mathbf{V}$ contains a set of orthogonal eigenvectors corresponding to $\mathbf{S}^T\mathbf{S}$, and $\Sigma^T\Sigma$ is a diagonal matrix containing the eigenvalues (or the squared singular values) corresponding to $\mathbf{S}^T\mathbf{S}$. A similar proof can be done to show that $\mathbf{U}$ contains a set of orthogonal eigenvectors corresponding to $\mathbf{S}\mathbf{S}^T$.

Using this relationship of eigenvectors and eigenvalues is important, as it is one way that the SVD of a matrix can be calculated- First by finding $\mathbf{S}^T\mathbf{S}$ and $\mathbf{S}\mathbf{S}^T$, then finding the corresponding eigenvalues and eigenvectors for each, and finally using those values to construct the matrices $\mathbf{U}$, Σ, and $\mathbf{V}^T$

2.3 Bayesian Correspondence Detection

Bayesian statistics is based upon updating current beliefs with new data. It provides a series of useful mathematical tools for dealing with formalized uncertainty, and ultimately reducing it. In our project, we use a form of Bayes Theorem to deal with uncertainty in measurement, and detect correspondence between newly scanned points and previously seen points. Formally, Bayes theorem is:

$$\Pr(H \mid D) \propto \Pr(D \mid H)\Pr(H)$$

Which in plain language reads as, "The probability of hypothesis H, given data D, is proportional to the probability of D, given H, multiplied by the initial probability of H". Qualitatively, this equation is taking the initial probability of H, $\Pr(H)$, and updating it according to the data D that is gained. This theorem as it is stated above is only really useful for testing a discrete set of hypotheses. To generalize over a continuous spectrum of infinite hypothesis, probability densities are required. Here forward, probabilities will be represented by Pr(), and probability densities will be represented by p(). The formal definition of Bayes theorem for probability density is as follows:

$$\mathrm{p}(H \mid D) \propto \Pr(D \mid H)\,\mathrm{p}(H)$$

A probability density is closely related to probability, but the two quantities are not equivalent. A probability density is a continuous function that maps out the relative likelihood, and not the probability, for a continuous set of hypotheses. In order to obtain an actual probability from a probability density function, the sum of probability densities needs to be summed up over an interval of hypotheses. The resulting sum is the probability of not one hypothesis, but the range of hypotheses that was summed. The mathematical formalization of this is as follows:

$$\Pr(H_a \leq H \leq H_b) = \int_b^a \Pr(D \mid H)\,\mathrm{p}(H)dH$$

In plain language, it reads "The probability that the so called 'True' hypothesis H is within the interval of hypotheses H_a to H_b is equal to the integral of the updated probability density function evaluated from b to a". If the interval between a and b represents the set of all possible hypotheses, the probability density function will, and must, evaluate to one when integrated over this interval. This is consistent with intuition: the probability that the 'True' hypothesis lies within the the set of all possible hypotheses is one.

It is important to note that it is impossible to find the probability of a singular hypothesis, because given a system with any non-zero amount of uncertainty, the probability that any one hypothesis out of a continuous set of hypotheses is correct is zero.

2.4 Iterative Closest Point (ICP)

Given two sets of corresponding points expressed in different coordinate frames, the Iterative Closest Point algorithm is used to find a set of matrix transformations that can be applied to one of the coordinate frames as to match the corresponding points in a common frame. It can be thought of as an optimization problem whereby the mean squared error (in this case, euclidean distance) between corresponding points in a set of points is minimized by applying one rotation and one translation. Given perfect correspondence, this optimization problem can be solved in closed form. However, given some amount of noise between corresponding points, this algorithm can, and should be applied multiple times to reach an optimized state, hence the "Iterative" part of the name. This algorithm is used in the project to translate points from the local to the global frame in a way that minimizes error.

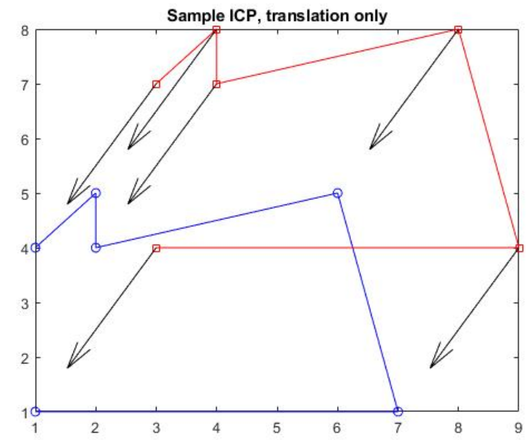

Figure 1: *ICP transforms corresponding points such that they match as best as possible.*

Qualitatively, the algorithm works by first expressing each dataset with respect to the average point position of said dataset. This is useful, as it effectively gets rid of any translation difference between the datasets and reduces the problem to involving purely rotation. Next, the algorithm solves for the rotation matrix that best aligns the points by looking at the singular value decomposition (SVD) of the covariance matrix between the two points. This operation is useful, as the covariance matrix can be thought of a transformation that transforms one set of points and the SVD can be used to express this transformation as a series of transformations (rotation, scaling, and then rotation again).

Now that the groundwork has been laid out, the specifics of the algorithm will be explained. Assuming a set of 2D points p expressed in the global coordinate system, and a set of corresponding points q expressed in a different coordinate system, such that:

$$p = \begin{bmatrix} x_{p1}, x_{p2} \dots x_{pi} \dots x_{pn} \\ y_{p1}, y_{p2} \dots y_{pi} \dots y_{pn} \end{bmatrix}$$

$$q = \begin{bmatrix} x_{q1}, x_{q2} \dots x_{qi} \dots x_{qn} \\ y_{q1}, y_{q2} \dots y_{qi} \dots y_{qn} \end{bmatrix}$$

We wish to find a rotation matrix $\mathbf{R}$ and translation vector $\mathbf{T}$, such that:

$$p = \mathbf{R}q + \mathbf{T} \tag{6}$$

It is likely that there is not perfect correspondence between points, and if this is the case, it is impossible to line up the two data sets perfectly. As such, (6) is re-framed as an optimization problem, like so:

$$\arg\min_{\mathbf{R},\mathbf{T}} \sum_{i=1}^{n} \|\mathbf{R}q + \mathbf{T} - p\|^2 \tag{7}$$

This expression is essentially minimizing the unnormalized mean square error (in other words, the total squared error) between the set of points p and the set of points q after applying the transformations $\mathbf{R}$ and $\mathbf{T}$.

Finding an expression for $\mathbf{T}$ is rather straightforward. Assuming that $\mathbf{R}$ is known and fixed, that is to say, the datasets p and q share a common orientation, the optimal translation is simply one that matches the centroids (or average point location) of the datasets. This can be formally proven by assuming $\mathbf{R}$ is constant, and setting the derivative of (7) with respect to $\mathbf{T}$ to zero in order to solve for the local (and in this case, global) minimum. The final expression for $\mathbf{T}$ is:

$$\mathbf{T} = \bar{p} - \mathbf{R}\bar{q}$$

Where:

$$\bar{p} = \begin{bmatrix} x_{p1} - \mu_{px}, x_{p2} - \mu_{px} \dots x_{pn} - \mu_{px} \\ y_{p1} - \mu_{py}, y_{p2} - \mu_{py} \dots y_{pn} - \mu_{py} \end{bmatrix}$$

$$\bar{q} = \begin{bmatrix} x_{q1} - \mu_{qx}, x_{q2} - \mu_{qx} \dots x_{qn} - \mu_{qx} \\ y_{q1} - \mu_{qy}, y_{q2} - \mu_{qy} \dots y_{qn} - \mu_{qy} \end{bmatrix}$$

Where μ is the average x or y position across all points in a dataset.

Calculating $\mathbf{R}$ will continue to use the mean-centered datasets $\bar{p}$ and $\bar{q}$. This is due to the fact that regardless of the datasets' positions and orientations relative to each other, the distance between points within a dataset, and the mean point of that dataset should be more or less consistent across corresponding points from the other dataset. We wish to minimize the error with respect to $\mathbf{R}$.

$$\arg\min_{\mathbf{R}} \sum_{i=1}^{n} \|\bar{p} - \mathbf{R}\bar{q}\|^2 \tag{8}$$

As shown in [1], with sufficient algebraic manipulation, (9) can be derived from (8).

$$\arg\max_{R} \operatorname{Tr}(\bar{p}^T \mathbf{R}\bar{q}) \tag{9}$$

Using a useful property of the trace:

$$\operatorname{Tr}(AB) = \operatorname{Tr}(BA) \tag{10}$$

for appropriately sized A and B, (9) can be rewritten as so:

$$\arg\max_{R} \operatorname{Tr}((\bar{p}^T)(\mathbf{R}\bar{q})) = \arg\max_{R} \operatorname{Tr}(\mathbf{R}\bar{q}\bar{p}^T) \tag{11}$$

The product of $\bar{q}$ and $\bar{p}^T$ is somewhat equivalent to an unnormalized covariance matrix–two identically sized sets of data that are mean centered are being compared in an element-wise fashion. In order to solve for $\mathbf{R}$, it is necessary to find the SVD for $\bar{q}\bar{p}^T$

$$\bar{q}\bar{p}^T = S = \mathbf{U}\Sigma\mathbf{V}^T$$

such that (11) becomes

$$\arg\max_R \operatorname{Tr}(\mathbf{R}\mathbf{U}\Sigma\mathbf{V}^T)$$

Using the identity described in (10) the following is obtained

$$\arg\max_R \operatorname{Tr}(\Sigma\mathbf{V}^T\mathbf{R}\mathbf{U}) \tag{12}$$

This is useful because $\mathbf{U}$, Σ, and $\mathbf{V}^T$ are all orthogonal. Because $\mathbf{R}$ is a rotation matrix, it is also orthogonal. Further still, $\mathbf{U}$, $\mathbf{V}^T$, and $\mathbf{R}$ are orthonormal, as they are purely rotation matrices. Because the product of orthonormal matrices is another orthonormal matrix, it is known that $\mathbf{V}^T\mathbf{R}\mathbf{U}$ is also orthonormal. The identity matrix is the orthonormal matrix that will maximize the value of the trace, as the only non-zero values are on the diagonal. With this in mind, it is now easy to solve for $\mathbf{R}$,

$$\begin{aligned} \mathbf{V}^T\mathbf{R}\mathbf{U} &= I \\ \mathbf{V}(\mathbf{V}^T\mathbf{R}\mathbf{U})\mathbf{U}^T &= \mathbf{V}\mathbf{U}^T \\ \mathbf{R} &= \mathbf{V}\mathbf{U}^T \end{aligned}$$

3 Quantitative Calculation

All of the mathematical concepts laid out in the previous section hold importance to the project as they are integral to performing SLAM (Simultaneous Localization and Mapping), which is the challenge of constructing and updating a map of an unknown environment while keeping track of the agent's position within the environment. In this project, the agent is a NEATO robotic vacuum cleaner equipped with a LIDAR scanner. For reference, the LIDAR returns a set of 360 radially even polar coordinates relative to the center of the LIDAR unit.

3.1 Correspondence

The Iterative Closest Point algorithm was employed to do a finer, and more accurate correction between the points from a new scan and the set of known points in the global frame. For this algorithm to work however, it needs to be fed points that are strictly correspondent. Therefore, potential cones that are detected in a new scan need to identified as either an old cone that has been seen before, or a brand new cone.

Because correspondence between points needs to be determined before ICP can work, it is very helpful to have a rough idea of where points from a new scan lie in the global frame. This is the reason that the new scan points were first initially transformed based upon known estimated position information. Transforming the points based on a position estimate derived from wheel velocities and time may not be good enough for multiple scans in a row, but for one scan, it will get the new scan points sufficiently close to their actual positions in the global frame, which will help with determining correspondence.

It is important to note that even though a small amount of dead reckoning based off of motor velocities was used, error should not accumulate over time. This is due to the fact that the final determination of the position of any new point in the global frame is handled by ICP on LIDAR points.

Initially, a very simple method for detecting correspondence was used. After transforming the new scan points using encoder information, each new scan point was paired with the closest known/visited point. If the distance d between two points was within an experimentally determined range of the mean distance μ between unique points ($d < 0.4\mu$), the point pair was labeled as corresponding. If d was outside of this range, the point was labeled as a new point. Once rough correspondence was determined, corresponding points were selected and fed in the ICP algorithm. All points within the new scan (both new points and points that corresponded to old ones) where then transformed based on the results of ICP, and then added to the global map of points.

Because determining correspondence is a very important part of being able to perform SLAM, a method using Bayesian statistics was developed and employed in order to determine correspondence in a mathematically formal way. Given the data r, an (x,y) point from the LIDAR, two probabilities needed to be determined: the probability of the new LIDAR data corresponding to a old point (H_{old}) and the probability of the new LIDAR data corresponding to a new point (H_{new}). Formally:

$$\begin{aligned} \Pr(H_{new} \mid r) &\propto \Pr(r \mid H_{new}) \Pr(H_{new}) \\ \Pr(H_{old} \mid r) &\propto \Pr(r \mid H_{old}) \Pr(H_{old}) \end{aligned}$$

$\Pr(H_{new}|r)$ and $\Pr(H_{old}|r)$ are mutually exclusive and completely exhaustive–A given data point r can either be a new cone, or it can be an old cone, assuming that there are no points that are falsely labeled as cones. Likewise, $\Pr(H_{new})$ and $\Pr(H_{old})$ are also mutually exclusive and completely exhaustive. this gives a useful property, as:

$$\Pr(H_{new}) = 1 - \Pr(H_{old}) \tag{13}$$

$$\Pr(H_{new} \mid r) = 1 - \Pr(H_{old} \mid r) \tag{14}$$

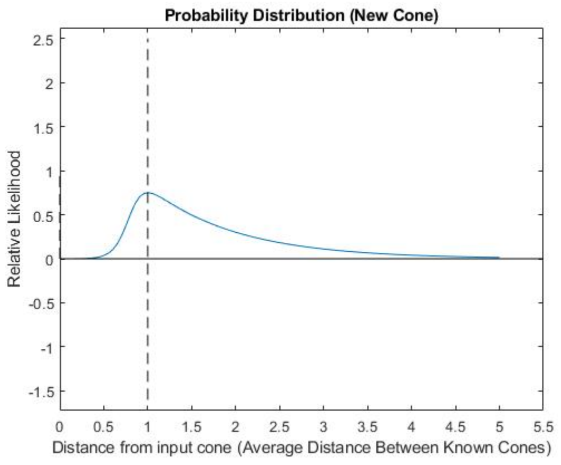

Figure 2: *The curve models the likelihood that a given point is a new point, given the relative distance (in terms of the average distance) to a given known point.*

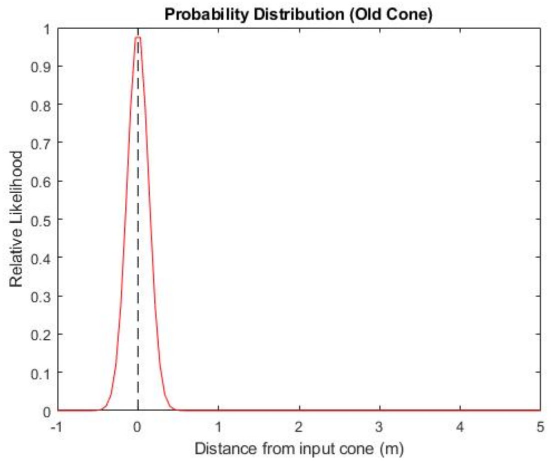

Figure 3: *The curve models the likelihood that a given point is a point that has been seen before, given the distance away from a given point.*

Staring with $\Pr(H_{old})$ and $\Pr(H_{new})$ is easy. They are roughly 0.75 and 0.25, respectively. In a typical scan, there are usually four points visible to the NEATO, and if the distance between subsequent scans is roughly the distance between cones, it is a reasonable approximation of the actual probability.

$\Pr(r|H_{old})$ and $\Pr(r|H_{new})$ are a bit harder. These probabilities require an estimation of the probability of a cone being in *any* location given it is either a new cone, or an old one. This is where probability densities are useful, as they allow for the testing for an infinite continuum of hypotheses.

$$\Pr(r \mid H_{new}) = \int_{ro}^{rf} \mathrm{p}(r \mid H_{new}) dr$$

where ro and rf represent the region that the point is expected to be in, which is essentially the uncertainty in the measurement. The set of all possible r's represent a probability density surface, which would contain the relative likelihood, or probability density, of a new cone being found for all points in 2D space.

A probability density function was chosen to represent the likelihood of new cones given knowledge of existing cones. The following base function, whose shape can be seen in Figure 2, was chosen for new cones:

$$\frac{e^{r-A}}{e^{8(r-A)}+1}$$

This curve qualitatively meets the expectations for the probability of a new cone around a given cone. There is a rather low probability very close to the original cone, a peak at the mean cone distance, and a gradual trail-off. This is the reason behind the constant 8. It effectively shapes the graph to have a more desirable shape. A is a constant that is the center of a cone. Its purpose is to center the function correctly around said cone. To get the probability density for any 2D point in space, given all of the known cones, the following is used

$$p(r) = \sum_{i=1}^{n} \frac{e^{r-A_i}}{e^{8(r-A_i)}+1}$$

In this case, A_i is the coordinate of the ith known cone. It effectively shifts the function to be centered around each cone in the known set of cones. Effectively, this creates a 3D surface, where each x and y coordinate has some associated z value that represents the probability density or relative likelihood of a new cone lying at that coordinate. A similar surface can be made to represent the relative likelihood that a point corresponds to an old point by simply defining a new probability density function. The following form, whose shape can be seen in Figure 3, was chosen:

$$e^{\frac{-(x-A)^2}{C}}$$

Where A is the position of a known cone, and C is a constant that represents the standard deviation or effective uncertainty in the measurement. This function was chosen as it also qualitatively meets the expectations for the probability of an old cone around a given known cone. The closer a point is to an known cone, the more likely it is to actually be that known cone.

With these functions, it is possible to find quantities that are proportional to the posterior probabilities $\Pr(H_{new} \mid r)$ and $\Pr(H_{old} \mid r)$, which matter most. In order to obtain the complete and normalized posterior probabilities, the property in (14) can be leveraged to normalize each probability on the same scale from 0 to 1.

For a given position of the NEATO, detailed in Figure 4, probability density surfaces can be generated. In this case, the NEATO was able to successfully detect and label three out of the four cones in its proximity, missing the cone that it is closest to, cone number 3.

Based off of this, it generated probability density surfaces in order to classify detected cones, as shown in Figures 5 and 6.

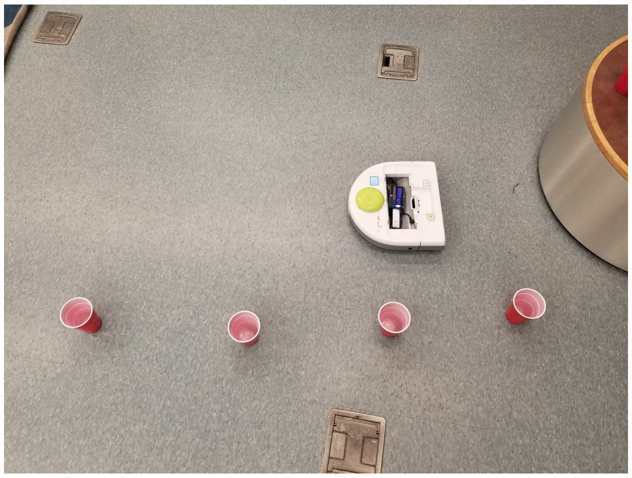

Figure 4: *The physical setup of a NEATO taking a test scan. In the scan, it labels three out of the four cones correctly, missing the cone directly next to the NEATO*

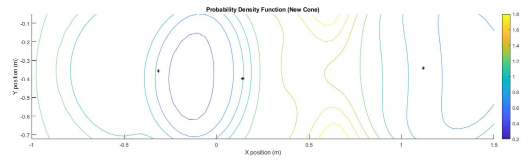

Figure 5: *The contours associated with the relative likelihood of a new cone. The highest probabiliy area is right where the missing cone should be–in between the 2nd and 4th cone.*

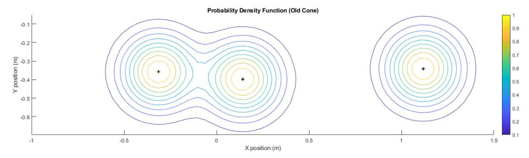

Figure 6: *The contours associated with the relative likelihood of an old cone. The highest probability areas are right on top of the known positions of cones*

Any new scanned cone center would be measured against each probability density surface, integrated over a small area to obtain probability, and normalized based on the prior and total probabilities. If the probability that the newly scanned point corresponded with an old point was greater than 50 percent, it was labeled as an old point, and if not, the newly scanned point was labeled as a new, previously unseen point.

3.2 Iterative Closest Point

Iterative Closest Point (ICP) was a really important tool in the scheme of this project. Once correspondence was detected between newly scanned points and previously visited points, ICP could be performed on the set of corresponding points to generate a transformation that would best align the new data with the preexisting data. Actually implementing the algorithm is straightforward, and as such, went without much of a problem. That being said, through a lot of testing, it was determined that ICP could be used for more than just a final reconciliation between the roughly transformed points and the global points. For some cases, the LIDAR cone centers that were returned from the cluster detection algorithm were too far away from existing cones to trigger a correspondence flag, even if the points actually did correspond. This would result in two points being very close to each other after ICP was performed on correctly identified corresponding points. In order to mitigate this problem, it was decided to perform the operation of correspondence detection and ICP multiple times. This would allow for several refinement transformations to be made, moving the points from the new scan ever closer to their global frame counter-parts in the hopes of discovering new correspondences, which would increase the robustness of the global map.

The workflow between ICP and the Bayesian Correspondence Detector was as follows:

1. Detect correspondence between list of known points and list of points from the new scan
2. Use ICP on corresponding points to get a transformation
3. Transform the *all* points from the new scan according to the transformation obtained from ICP
4. Repeat from step 1

In the final implementation of the code, it was determined that three iterations of correspondence detection was sufficient in capturing any stray correspondences that were initially missed.

4 Implementation

Apart from ICP and Bayesian Correspondence, there were many other tools developed in the implementation of the project. Each of these additional tools had varying levels of mathematical intensity, but were all consistent in that they were less mathematically involved than the main two analytic methods. This section seeks to provide detail and light mathematical explanation behind these additional tools, as well as give context to the tool in the larger view of the project.

4.1 Transformation Matrices and Reference Frames

One main usage of transformation matrices was to move the scanned points from the LIDAR sensor's reference frame to the NEATO's. As the origin for the NEATO's reference frame was defined as the center of its wheelbase and the LIDAR was mounted towards the rear of the NEATO, there was a difference between their origins as seen in Figure 7. A constant translation was applied to the scanned data to rectify this misalignment to represent the data points with respect to the NEATO's perspective. After this translation was applied to correct the data, the data could then be processed with ICP to be transferred into the global frame.

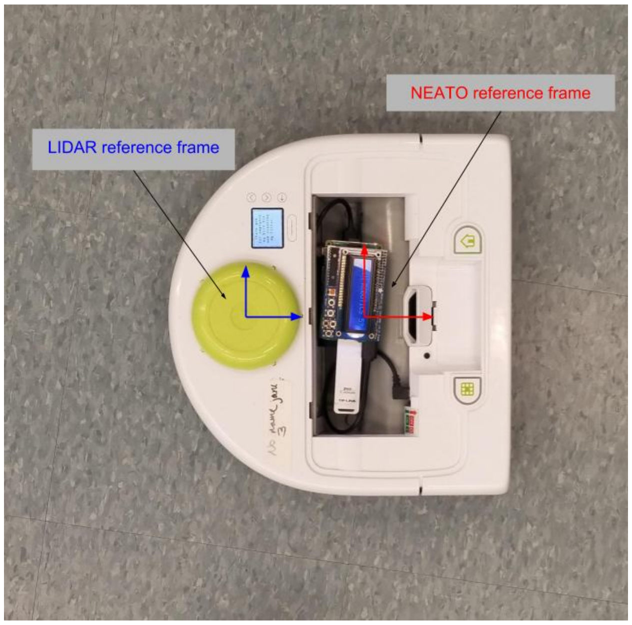

Figure 7: *The NEATO's LIDAR is mounted 0.1 meters behind the center of the NEATO's wheelbase. A translation of 0.1 meters in the x-direction is used to align the data scans from the LIDAR's reference frame to the NEATO's reference frame.*

4.2 Cluster Detection

Once the data points were corrected into the NEATO's reference frame, post processing in the form of cluster detection was used to extract the position of the path-defining cones. Cluster detection is the process of labeling subsets of data within a larger group of data. This is a pretty large and common problem in various different fields, and as such, can be looked at and framed in many different ways. Performing this step greatly reduces processing time and complexity, and as such, is highly desirable.

Because the points of interest are received originally from the LIDAR, they are already in an order of ascending θ, which makes cluster detection really simple. Detecting clusters is now as easy as looking at consecutive polar coordinates (ie: steadily increasing θ) and the difference in radius between them. Given a set of n points $\mathbf{p}$, consisting of row vectors θ and r containing angle and radius values for corresponding points:

$$\mathbf{p} = \begin{bmatrix} \theta \\ r \end{bmatrix}$$
$$\theta = [\theta_1, \theta_2, \theta_3 \dots \theta_n]$$
$$r = [r_1, r_2, r_3 \dots r_n]$$

μ, the average radial difference $(r_i - r_{i-1})$ between consecutive points can be computed by the following:

$$\mu = \sum_{i=2}^{n} \frac{r_i - r_{i-1}}{n-1}$$

The standard deviation σ of the radial difference between consecutive points can be computed by the following:

$$\sigma = \sum_{i=2}^{n} \frac{r_i - r_{i-1} - \mu}{n}$$

The standard deviation is a very useful metric to look at, as it shows how much variance is present within the data. By scaling σ by an experimentally determined constant C (in our case 0.15), it is possible to easily search for outliers in the radial difference between consecutive points. If the radial distance between consecutive points is greater than $C\sigma$, then the points can be flagged as the beginning and end of a new group.

Additional characterization needed to be applied to the clusters to identify individual clusters as cones. A circular interpolation method was added to increase the accuracy and refine the cone detection. The circular interpolation method analyzed and filtered objects in the environment based on their expressed curvature. The method chose three evenly spaced points within each cluster and used the perpendicular bisectors of the chords connecting the points to identify the center of the circle. The average distance between the calculated distance and the points in the cluster was evaluated. A threshold of 0.4 and 2 times the expected radius of the cones was set and if the average experimental radius of a cluster was within the threshold, the cluster was labeled as a cone and the center was used to define the position of the recognized cone.

Figures 8, 9, 10, 11 outline the different levels of filtration that the cluster detection algorithm performs.

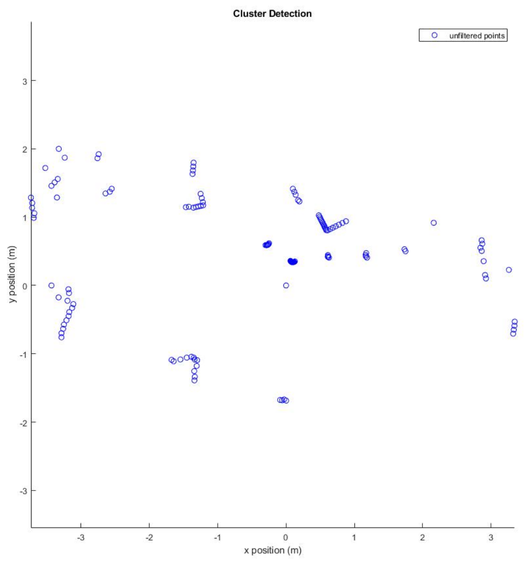

Figure 8: *This is Raw LIDAR Data, which is quite messy and includes a lot of noise extraneous to the system of that is being measured.*

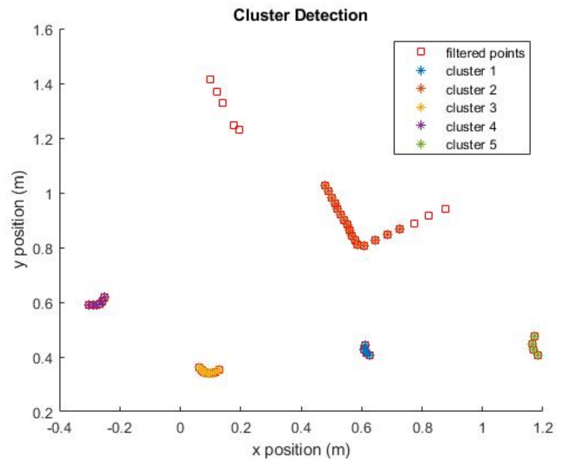

Figure 10: *Individual clusters are marked out based on the standard of deviation in radius between LIDAR points. Only clusters of 3 or more points are flagged and stored from this stage to the next.*

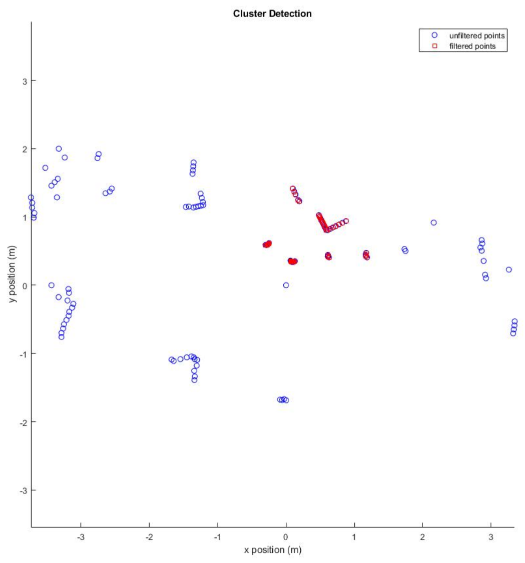

Figure 9: *Close points within 1.5 meters of the NEATO, here in red, are flagged for further analysis. Anything further away is deemed to be unsuitable for drawing any valuable conclusions.*

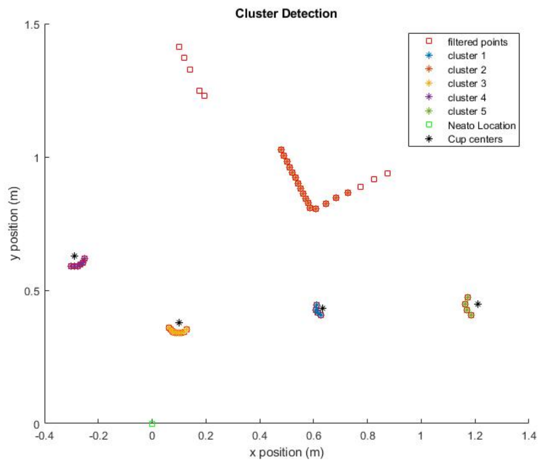

Figure 11: *From the clusters, individual cones are picked out based on curvature. Only the clusters that are close in shape to the cones get flagged and passed forward for Correspondence testing and ICP*

4.3 Path Generation and Movement

Having found the positions of the cones from the LIDAR scan, the next step was to generate a path for the NEATO to follow. While it would have been easy to simply send the NEATO to the position of the cone closest to it, it was necessary for the course to be preserved for future laps and therefore would have been impractical for the NEATO to have been constantly running over the cones. With this in mind, it was necessary to implement a method of generating a path based off of the positions of the guide cones but offset at a distance to allow the NEATO space to maneuver. This was achieved by first identifying the cone closest to the NEATO using simple euclidean distances. The closest cone was removed from the list of cones and as the cones were not returned in order, the next cone in line was found by first sorting the list of cones in order of ascending euclidean distances. From the sorted list, the dot product of the vector connecting each cone to the NEATO was calculated with the vector representing the NEATO's orientation to find the one that was closest that was also in a direction similar to the NEATO's orientation.

Once the two cones closest in order to the NEATO were identified, a vector was drawn between them and a new vector normal to it was drawn originating from the second cone. This new vector defined the direction of the offset and the magnitude determined the distance of the offset. The magnitude of the offset vector was set to 0.5 meters to allow the NEATO to closely follow the cones without running into them. A vector was drawn from the NEATO's current position to the new position to determine the path for the NEATO to achieve the new position.

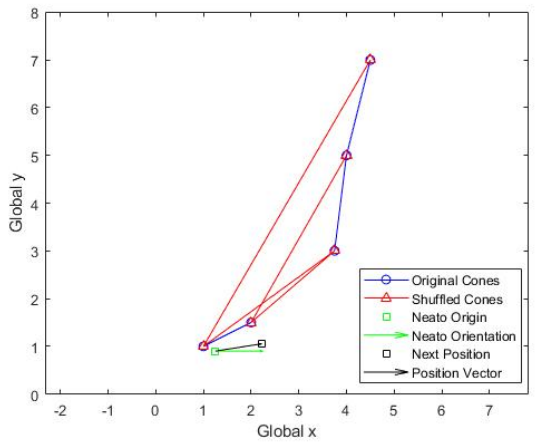

Figure 12: *An example of the offset path generated based on seen cones from LIDAR scan.*

Based on the path vector created, commands were written to convey this movement to the NEATO. With a differential drive, it was necessary to define the individual velocities of the NEATO's left and right wheels. For linear movement, the left and right wheels were both set to have a velocity of 0.1 m/s while for angular movement, one wheel was set to 0.5 m/s while the other was set to -0.5 m/s depending on the direction of the desired turn. As the linear velocity V of a differential drive is defined as the following:

$$V = \frac{V_L + V_R}{2}$$

the linear velocity of the NEATO was found to be 0.1 m/s. The angular velocity ω of a differential drive can be found using the following:

$$\omega = \frac{V_L - V_R}{d}$$

where d is the track width or distance between the left and right wheels. From this, the angular velocity of the NEATO was found to be ± 4.17 rad/s. Using these set velocities, the time needed for the movements was calculated by dividing the magnitude of the path vector by the linear velocity for time moving forward and dividing the angle between the NEATO's orientation and the path vector (in radians) by the angular velocity for the time spent turning. The velocities commands were then sent to the NEATO with the turning being performed first for the respective amounts of time. The changes in the NEATO's position and orientation were saved to perform rough alignment for the next scan.

5 Results and Discussion

For the final deliverable, a SLAM algorithm was developed to allow the NEATO to traverse an unknown course while simultaneously making an accurate map of said course. In developing this algorithm, each of the different techniques outlined in Sections 2, 3 and 4 was implemented such that each tool worked well with the others. Despite a large amount of time devoted into the testing and refinement of the different tools used in the project, in the end, the NEATO was never able to fully navigate and map out an extensive course. In the end, small errors in how the data was parsed and stored over multiple scans eventually accumulated over time, and threw the NEATO off. After roughly 13-15 transforms (a cycle consisting of NEATO movement and a collected scan), the theoretical model of the real world tended to diverge so much from the real world that the Bayesian Correspondence detector was unable to classify corresponding points correctly. On the one hand, the inability of the NEATO to traverse a full course is rather unfortunate. On the other hand, the performance of the NEATO was much better than it could have ever been simply relying on dead reckoning. Over the course of 13-15 transforms, the virtual map of the course generated by the NEATO different from reality by about 0.2 meters, whereas it is not uncommon for more than 0.2 of error to be introduced after just two transforms of NEATO when relying simply on dead reckoning techniques.

Figure 13 shows the final scan that the NEATO was able to produce on one of the most successful runs. In this case, the course it was trying to navigate was roughly a square whose edge length was about 3.5 meters, and whose corners were slightly rounded. Red points represent the known locations of cones in the global frames. Green asterisks are the positions of cones in the newest scan, corrected only based off of the velocity and time data sent to the NEATO. Also included is the probability density surfaces for both old and new cones associated with the points highlighted by the larger red points. The surface associated with the likelihood of an old cone has tight peaks around each known cone, and the surface associated with the likelihood of a new cone is much more spread out and less concentrated around known points.

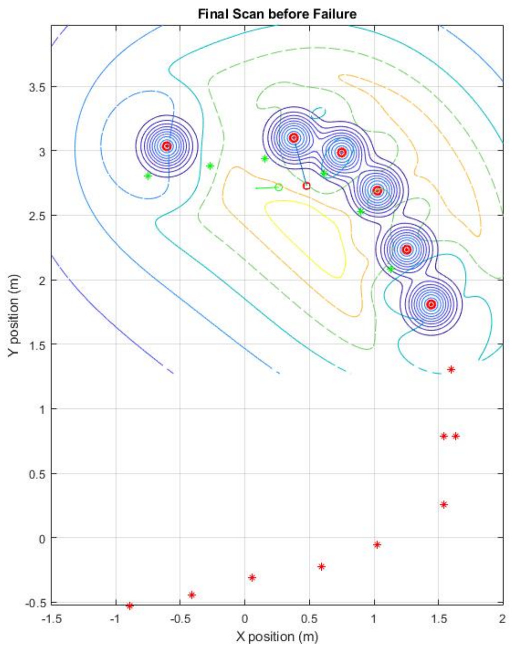

Figure 13: *The final scan that the NEATO was able to make before failing. In this case, failing was being unable to match up corresponding points, and being unable to proceed*

The reason behind the inevitable accumulated error between the virtual model of the course and physical course is due to how much importance is placed on points that have been seen before. When transferring new points into the global frame, new points are always transformed such that the new points that have correspondence line up almost perfectly with their corresponding point in the global frame. This is placing much too high of a value on the accuracy of the preexisting value. Each point is originally derived from the LIDAR scan passed through the same cluster detection algorithm, and as such, given no other information or data, should carry the same level of uncertainty with it. By moving only one set of points and keeping the other basically stationary, it is effectively treating the unmoving set of known points in the global frame as much more accurate than the set of new points, which is just not true.

6 Conclusion

A lot was learned over the course of this project. While it was not possible to achieve what was originally planned, a method that is much more robust than dead reckoning was developed to the point where a few more incremental refinements could make it possible to achieve what was originally planned. Perhaps the greatest takeaway from all of this is the importance of minimizing uncertainty and error wherever possible and wherever convenient. Over a series of many transformations, it is very difficult to keep track of the position of the NEATO while managing the error in position, so any reduction of uncertainty can and will help. Several changes have been identified as possible ways to reduce uncertainty and thus improve the performance of the project if there was more time.

6.1 Robust Reconciliation between Known points and corresponding New points

Every cone center that is derived by performing cluster detection on a LIDAR scan has some degree of uncertainty associated with it. While this uncertainty is small, especially compared to the encoder uncertainty, it is nonetheless non-zero. Seeing the same point across multiple scans will help to reduce this uncertainty. When updating the positions of cones in the global frame, there needs to be a way to reconcile information previously seen and new information that recognizes the inherent uncertainty in each of the data points. A method for accomplishing this has been devised with the following steps:

1. Detect correspondence between list of known points and list of points from the new scan

2. Use ICP on corresponding points to get a transformation

3. Transform *all* points including NEATO position from the new scan according to the transformation obtained from ICP

4. Repeat from step 1 until all correspondences are found

5. Update position of corresponding points to be the average position of *all correspondences ever found*

6. Use ICP on corresponding points from the new scan and the new averaged corresponding points to get a transformation

7. Transform *all* points including NEATO position from the new scan according to the transformation obtained from ICP

The method outlined above is essentially a crude implementation of a Kalman State Estimator, a piece of mathematics used to reconcile different measurements, each with its own uncertainty. This is a method that was considered for implementation, but was eventually scrapped due to time concerns.

6.2 Clustering based on Euclidian distance

When performing cluster detection, the radial distance between neighboring polar points is evaluated when calculating the mean and standard deviation in the distance between two points. This is a pretty reasonable approximation that appeared to perform well. However, a more accurate method would be to calculate the actual Euclidean distance between neighboring points. This is a simple and somewhat trivial problem to fix, but is nonetheless a source of error when identifying points.

6.3 K-Means clustering

An additional clustering algorithm is the K-Means Clustering method. Given a set of points X, a set of k centroids are randomly picked. In each iteration, individual points are assigned to the centroid that is closest to it. The average position of the final group is computed, and the corresponding centroid is moved to that point. Over a few iterations, the algorithm quickly places each centroid at the average position of discrete groups. The issue with K-Means Clustering is that k needs to be known beforehand, which is impractical because it is impossible to know the number of visible cones before scanning. Luckily because the algorithm is quite fast, it can be run over a series of potential k values, and the one that minimizes the total variation within each cluster without creating excessively small groups can be chosen.

The authors of this paper are not at this time sure whether this method would be strictly better than what is currently used. However, K-Means Clustering is a very well known and established clustering algorithm that is widely used in fields such as machine learning. This suggests that it can be adapted to this problem with performance comparable to the algorithm currently used.

6.4 Encoder-based Odometry

When performing the initial transformation that roughly transforms points in the NEATO frame to the global frame, theoretical target distances and angular displacements are used. This assumes that the NEATO is able to somewhat reasonably accurately carry out commands sent to it. When the NEATO's movement is based completely on velocity and time, this is just simply not a good assumption to make. As such, it would be much more accurate translate and rotate the representation of the NEATO based on numbers specified by encoder values returned after making a move.

7 Bibliography

[1] O. Sorkine-Hornung and M. Rabinovich, "Least-Squares Rigid Motion Using SVD," p. 5. [2] H. Cho, E. K. Kim, and S. Kim, "Indoor SLAM application using geometric and ICP matching methods based on line features," Robotics and Autonomous Systems, vol. 100, pp. 206–224, Feb. 2018.

8 Appendix

All code written and used in our NEATO implementation can be found on Github.