Scanning and Reconstruction (SCAR)

Jamie Cho, Benjamin Ziemann

December 27, 2018

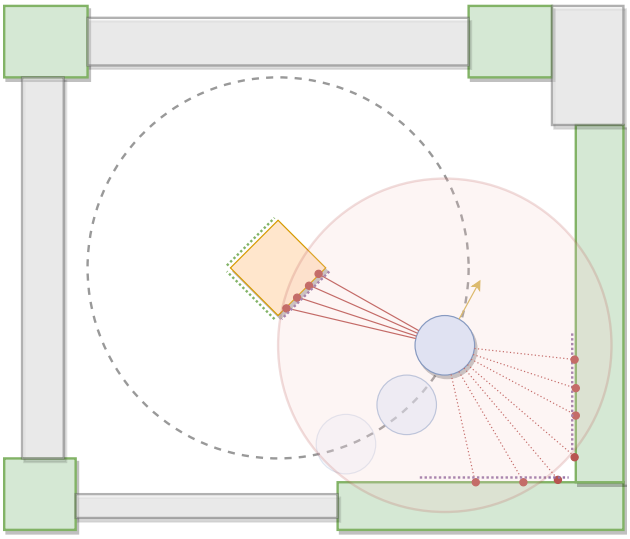


Figure 1: Abstract representation of the reconstruction process.

All code referenced in this document and more can be found in the project's Github Repository.

1 Overview

The SCAR (SCanning And Reconstruction) project is an attempt to use the Neato as a 3D scanner by driving around an object and using the LIDAR and camera to recreate it digitally. We are treating this as a simpler, though still well within scope, implementation of a SLAM algorithm, operating with the constraint of an isolated and deterministic environment.

The overall system is a pipeline which applies transformations to a set of accumulated points and current points. In the development of this project we encountered several routes, some of which we explored in depth, some of which we did not, for each major component of the system. These other solutions, along with the solutions we went with, are included in depth in each components section below.

Overall, our system architecture is described in Fig. 2. After an initial map consisting of the first several scans with the essentially stationary robot is generated, raycasting and ICP begin to be performed. Raycasting limits points already on the map to physically viable points based

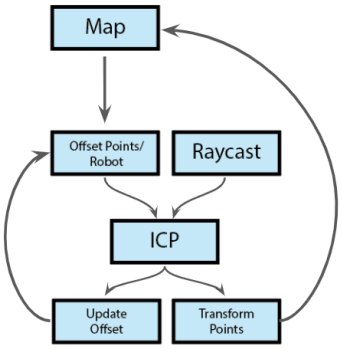


Figure 2: Overview of our system architecture.

on the robot's positon; ICP then takes that reduced map and fits the current scan points to it. The resulting transformation is applied to the current scan points, which are in turn added to the ever growing map. This rectifying transformation is also used to update known map-to-odom coordinate offsets, giving us a headstart on correcting the robot's position estimate and the scan point locations in the next iteration of the cycle.

2 Scan

Using the onboard LIDAR, we are able to collect points, coordinates of environment features around the Neato to use to build up our mapping to better orient ourselves in relation to the object we are "scanning". Using just the scans, as seen in the drifting behaviour of RVIZ, points would incorrectly "Drift" with the robot as we turned and move, creating far too much of a trail to accurately make a map. This effect is most readily seen in Fig. 3, with significant rotational drift of the wall.

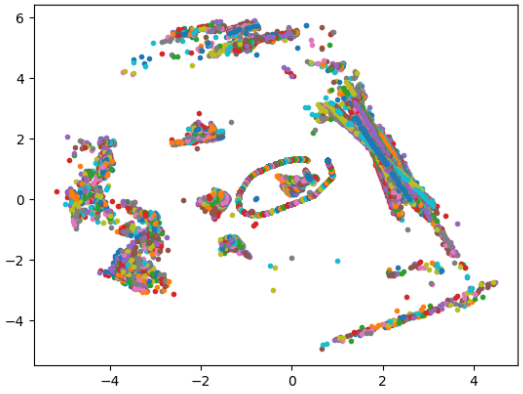


Figure 3: Raw points plotted of AC109. Noticeable oval is the trash can, situated in the center of the operating arena.

From the start we also would have liked to use the camera to both better orient ourselves in the environment as well as create a full shell of the object, not just a contour. We did not have time

to implement this but believe it would have been similar to the current setup, with a greater focus on building the object for camera output and environmental orientation using the LIDAR output.

One issue we faced was questioning whether or not the 5Hz-360 degree scans would be enough to accurately get dense enough points to fill out the map. This was abated as we implemented rounding to a given resolution for points as well it providing good enough scans for our purposes; while substantially more rigorous in implementation, cartographer[2] authors report compelling mapping performance at the Deutsch museum with a 2Hz lidar.

3 Offset

In accordance how the ROS architectures resolve positional drift over time, the robot's continuous position with respect to the starting location was represented with the *odom* frame. At each scan-matching steps, we estimated the drift between the robot's wheel-encoder based estimated position and the absolute position and compensated for the error through the *map* frame, which would represent the discontinuous, but *true* position of the robot at that instance, consistent over time.

4 Map

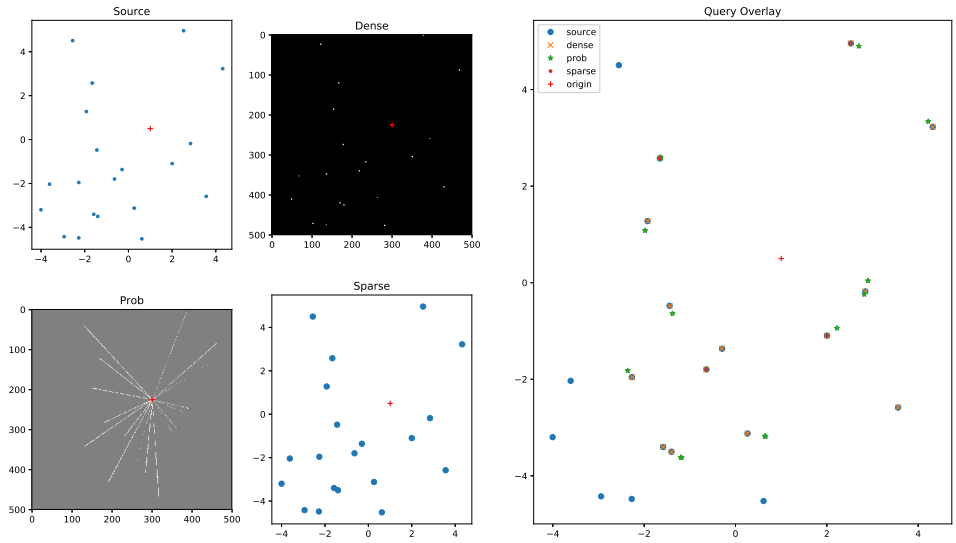


Figure 4: Stylistic comparison and characterization of different map variants.

We have explored several approaches to the mapping problem, where each occupancy information will be stored as sparse or dense representation; the intuition for the sparse representation is that only a small portion of operating environment is occupied, with most points being free space. In a relatively small environment, the sparse representation would provide both efficiency of data storage and faster queries.

In the sparse architecture, the map of accumulated points works as a dictionary, where keys are coordinate tuples and values were how many times a given point had been seen. After being seen a certain number of times, the point would be permanently added to the visualization of the robot's environment and be used (given it was possible to scan) to align future points. Resolutions were used to control how much points were clumped together, enabling us to get more precise maps although with some trade-offs (see Raycast section).

After exploring the threshold-based approach, we determined a more statistically driven model where the map would store log-odds of occupancy rather than observation counts, which would more readily be converted into probabilities (admittedly with some arbitrary parameter selection). Similar in set up to what we ended up going with, this approach would increase the probability of a certain point actually existing each time it was scanned and decrease it if it was both able to be seen by LIDAR but not, along with factoring in the error of a LIDAR missing a point.

One important additional step that was introduced in the probabilistic map was the incorporation of the inverse sensor model, where the physical characteristics of the beam was accounted for as the map was updated and queried.

4.1 Lidar Characterization

In order to utilize the probabilistic model effectively, it was imperative that we had an understanding of the lidar characteristics. According to the official specifications, the technical capabilities of the lidar is as follows:

Parameter	Value
Wavelength	785 nm
Pulse Freq	1.8 kHz
Pulse Duration	200 us
Beam Power	2.1 mW
Beam Diameter	2.3 mm
Beam Divergence	-1.5 mrad

Table 1: Neato XV series lidar specifications.

Accordingly, the expected broadcast-beam intersection from a given point is as follows:

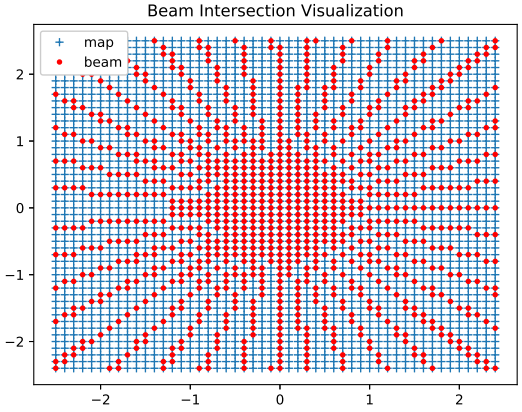


Figure 5: Cell-Beam occupancy visualization based on physical lidar characteristics.

Both of the approaches required us to better refine the map however has we found that when trying to align points to the built up map it would try to fit the current scan to places it couldn't possibly see in its current location but had in the past. In response, we added pseudo-raycasting to deal with these obscured points.

5 Raycast

In order to improve the stability of the ICP algorithm as it computed the correspondences between the registered map and the incoming scans, we implemented a simple version of the ray-casting algorithm which would only return the "visible" portion of the current map with respect to the robot's position, without corrupting the input with irrelevant points that should not be accounted for when computing the transforms.

Challenges we faced here included resolution tuning. If the points were clumped too sparsely, scans may still make it through, creating disjointed blocks of points for the same object. Too dense though and recognizable features of the visualized map decayed to amorphous blobs of points.

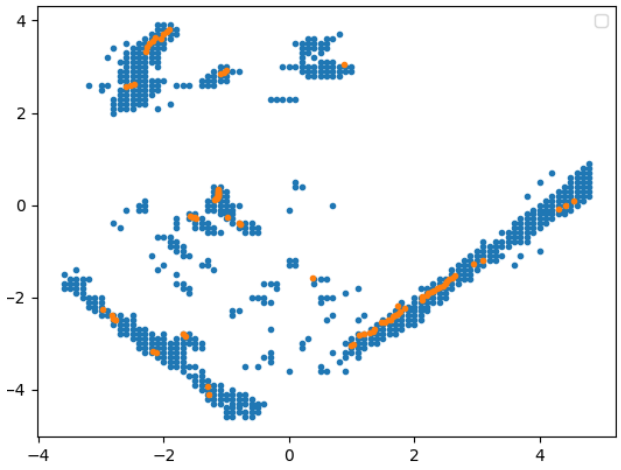


Figure 6: Points caused by noise build up around the actual object location lead to further incorrect offsets by ICP

6 Iterative Closest Points (ICP)

Iterative closest point is a least-squares based optimization algorithm that resolves transformations between two point-sets based on relaxed correspondences assumption. It works well with the initial assumption that the matching points are relatively near each other ...

In many ways, ICP was a natural choice in our situation, as many of its fundamental assumptions were true, including: - a relatively large set of correspondences with well-seeded search correspondences

Whereas we started with an existing bare-bones implementation of ICP from a Github Repository, we improved the original algorithm throughout the project to make it better suited for our particular use-case. In particular, we implemented a point-to-plane-based error metric based on

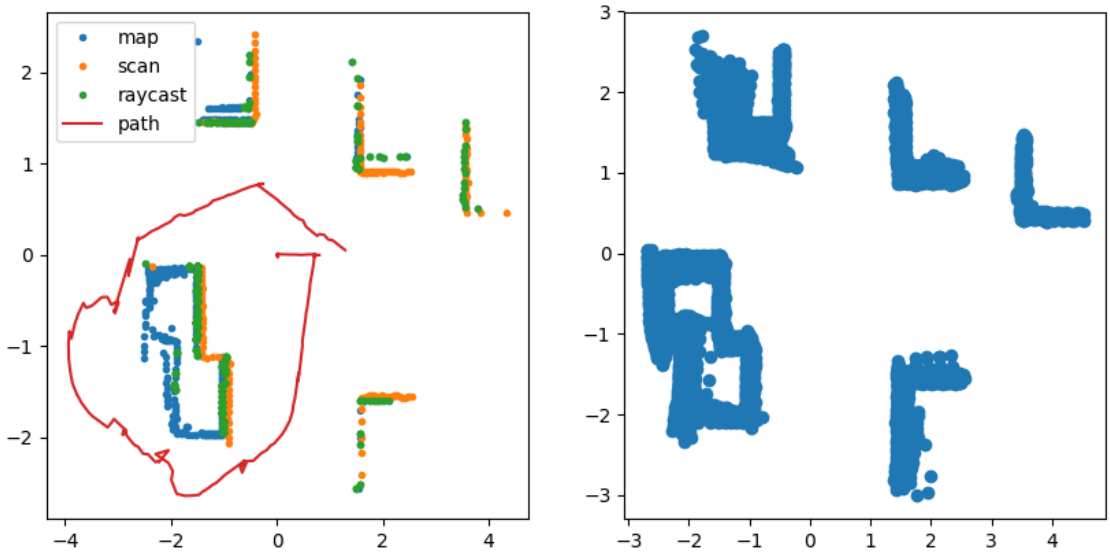


Figure 7: Raycasting from the center of the blocks. Points on the far side of the blocks are ignored (no green)

the steps listed here. In practice, we haven't observed a significant improvement of performance, which may be due to the relatively few feature points as input to the algorithm, poor normal-vector estimation, etc.

The output of a simple 2D homogeneous transformation matrix also worked well for our application as we could easily apply it to existing arrays of points with minimal preparation or reworking the data.

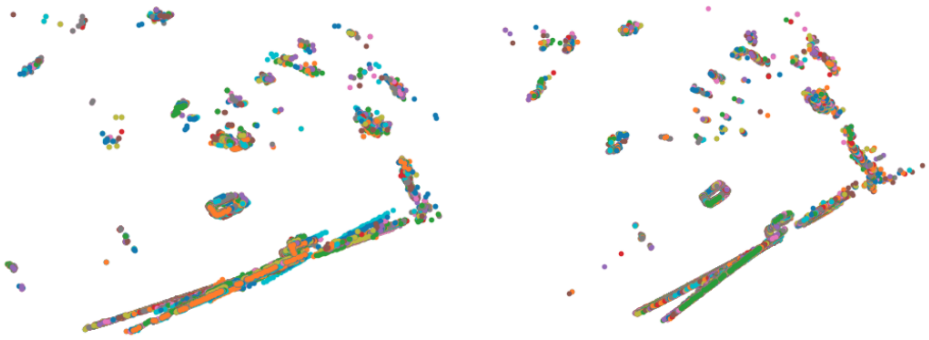


Figure 8: Raw points (left) vs First pass of ICP implementation (right)

6.1 Point-To-Plane Error Metric

One intuition behind icp-based correction is that surface-features can only provide corrections in the direction orthogonal to such features; i.e. when computing the correspondences, certain features must allow *sliding* behavior for transforms to be rectified correctly.

Accordingly, we implemented an advanced version of the ICP that estimates *better* transform

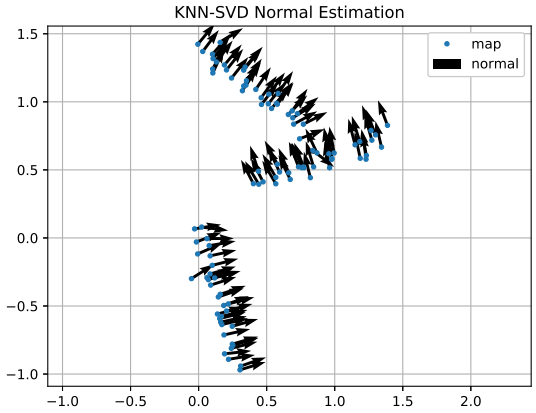


Figure 9: SVD-based Normal Vector Estimation operating on K-Nearest Neighbor where K=20.

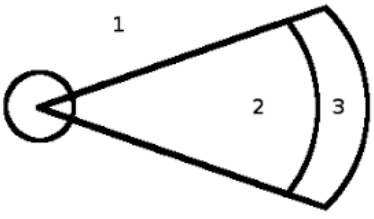
estimates with the point-to-plane error metric, with the normal vector estimated through the least-signifcant eigenvector found through SVD on the 20 nearest neighbors around the point.

Other explorations include refinement of final transforms and RANSAC-based inlier filtering for offset transforms based on the implementation of *estimateAffine2D()* routine provided in OpenCV.

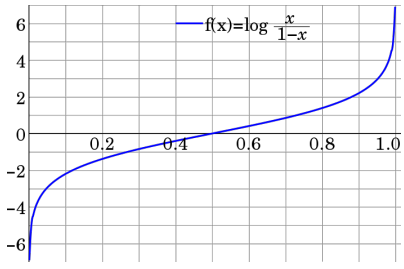
7 Update Map

As the pure odometry-based localization produced low-quality scan results due to drift over time (see Fig X (in Scan)) , it was necessary to correct for the robot's absolute position through tracking local features and producing a map against which the map would be localized. In essence, the mapping steps were approached with a SLAM (Simultaneous Localization And Mapping)-like architecture, where each observation was coupled with prior beliefs to produce both a higher fidelity map and a more accurate estimate of position.

One initial challenge that we faced was that, with the architecture at the time, there was no way to *refine* the map with incoming scans; i.e. *stale* points could not be cleared. We incorporated a similar approach to the elementary ray-casting method as described above in order to clear the points that should no longer be present in the map.



(a) The inverse sensor model

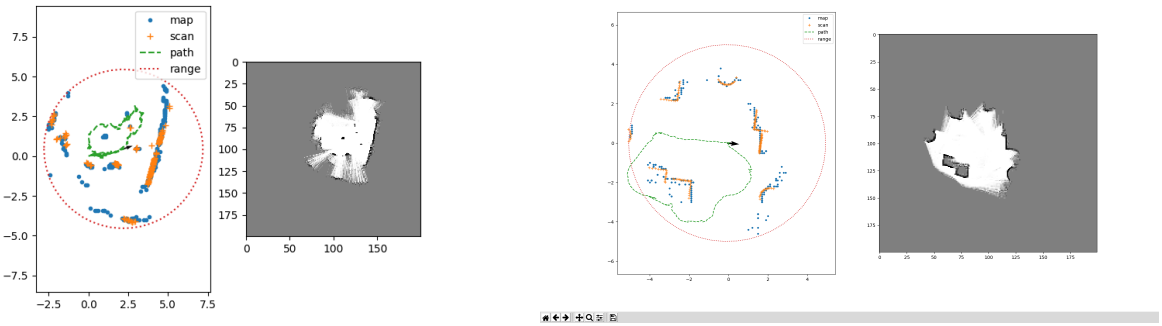


(b) The Logit Function

Figure 10: Improvements on map updates and representations; clearing and adding logic is jointly handled with the inverse sensor model with the logit representation.

The map update logic was re-structured with a similar ray-casting approach, where the beam parameters are directly incorporated to specify the area on the map that would be updated; i.e. the cells outside of the beam would be ignored, closer than the "hit" points would be cleared, and near the "hit" points would be adde, respectively, with logit-based probabilistic increments. This significantly improved both the map quality and the ICP queries down the pipeline.

8 Results



(a) Results from real-world run at AC109

(b) Results from Gazebo Simulation

Figure 11: Trajectory and map reconstruction results from both simulated and real-world runs.

As part of our learning goal involved effectively utilizing the simulation for our development, we report the scanning results based on experiments from both real and simulated environments.

While no quantitative assessment on the physical map is available, the visible landmarks on either map appear consistent with the operating environment in both cases, despite adversarial conditions such as sparse features and high drift in robot odometry estimates.

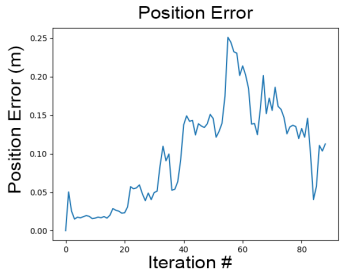


Figure 12: Error in position over time, controlled within 0.3 meters throughout the simulated run.

We also tracked the estimated position of the robot using the ICP transformation offsets vs the ground truth position from a Gazebo simulation. What we saw matched with the non-quanatative mappings of the error being fairly stable and minimal, although trending positive over time.

9 Reflections

9.1 Challenges

Throughout the implementation of our pipeline, we have faced a number of challenges including:

- Rectification of ICP-computed alignments
- Identification of feature-deprived regions
- Stable Sampling [1]
- Loop-closure

ICP would often compute *jumps* that are extreme changes in the transform offsets, which likely stemmed from incorrect estimate of correspondences and happened almost exclusively in feature-deprived conditions, which argues for identification of such scenarios; one proposed solution to identifying such conditions is the covariance-based stable sampling strategy as descrbied by Gelfand et al.[1].

Similarly, we anticipate a large boost in performance with submap construction and loop closure, which would unfortunately introduce significant computational cost. Since this was a fairly heavy task to undertake, we have not explored this option within our time-frame.

9.2 Improvements

We have had a lot of parameters embedded in the pipeline without rigorous characterization or justification. To name a few:

- RANSAC-based final transform refinement
- Transform "Jump" Filtering Threshold
- ICP Algorithm termination tolerance
- Map update confidence
- Map query threshold
- ICP Error Metric (point-to-plane) comparison

Many of these heuristics need a proper exploration into how the choice of the parameter impacts the output quality, as well as how they interact with each other in the pipeline.

However, the stochastic nature of the current pipeline makes it difficult to test each configuration in a robust and deterministic manner; furthermore, fine-tuning these parameters turned out to have diminishing returns compared to larger chunks of the task we were tackling at the time. We thusly identified this as an opportunity for future exploration rather than an immediate task to resolve.

Furthermore, there are multiple points in the pipeline where there exist significant benefits from performance optimization; a lot of computations are currently duplicated, such as nearest-neighbor estimation,

Whereas the isolated development of these modules enabled rapid code development, a definite opportunity for the future is to re-use intermediate structural representations of data in multiple steps of the pipeline to reduce the computational load.

References

[1] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, and Marc Levoy. Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3D Digital Imaging and Modeling (3DIM)*, October 2003.

[2] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.