Implementation and Testing Of Algorithms On Jackal Robot For Smart Indoor Navigation

MEng. Project Report
AER 1810

Aniket Sanjay Gujarathi

Student ID : 1007629979

Institute for Aersopace Studies
The University of Toronto

List of Figures

Contents

Chapter 1

Introduction

Transition from simulation to real-world is a huge step to push the research to applicability. While simulation has been well-established for integrated robot-software testing, such environments do not provide an exact representation of the issues faced while integrating the algorithms on real robots. Accounting for such issues in the existing software and tuning on the hardware is an important step to ensure smooth operation of robots. A failure to incorporate these may result in erroneous behaviour of the robot with the possibility of harming the people and its surroundings. Through this report, we will discuss some of the issues faced while setting up the robot - Clearpath Jackal UGV, and implementing a self-supervised approach for indoor navigation.

The aim of this project is to incorporate a self-supervised navigation pipeline on a robot and test the outcomes in a real-world environment. The navigation pipeline is based on the prior work of Hugues et.al. in [1], which predicts semantic labels without the need for tedious human annotation. Using multi-session SLAM and point ray-tracing algorithms, the system aims to annotate the maps in an automated manner, which is then used to train a network to predict classes of objects like walls, short-term movables, and long-term movables given a single lidar frame. Due to the novel multi-session automated annotation process as described in Fig. 1.1, the network improves itself over multiple runs. On simulated data, this system shows remarkable improvements on localization compared to AMCL and GMapping even in hard scenarios generated by changing the world parameters like adding a variety of actors exhibiting different motions, including furniture, etc.

In the extension of this work in [2], the authors suggest a novel lifelong navigation method by learning spatiotemporal occupancy grid maps. The navigation pipeline aims at providing an accurate prediction of the temporal evolution of the environment to enable the robot to plan and traverse the trajectories efficiently. This method depends on the self-supervised segmentation pipeline in [1] to generate the different classes based on moveable probabilities given a single lidar frame which is passed to a 3D-2D feedforward architecture to generate spatiotemporal occupancy gird maps (SOGM) of dynamic environments. These SOGMs are used to generate a safe and eficient plan using a time-elastic band based planner.

In this report, we discuss the setting up of the Jackal Robot to incorporate the navigation pipeline explained previously. We discuss some issues faced during the setup of the robot in Chapter 2, and proposed solutions for the same. In Chapter 3, we discuss the issues of motion distortion on the PointSLAM algorithm and our approach to solve this issue. Chapter 4 discusses the particular constraint of the multi-session SLAM approach to provide initial alignment in every run and suggest some solutions for the same.

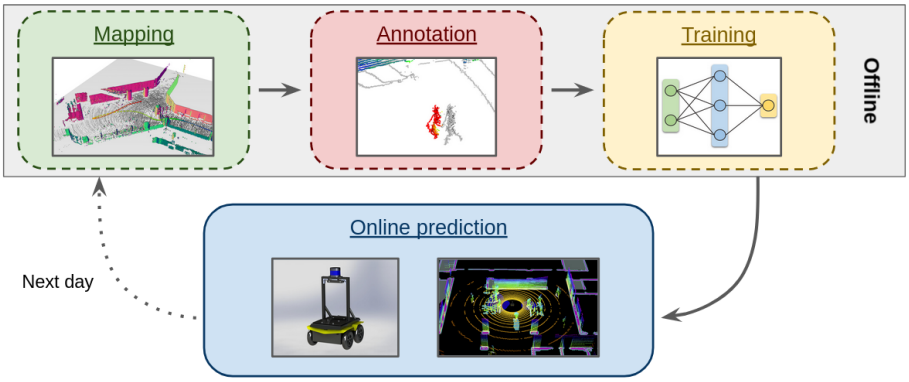

Figure 1.1: Self-Supervised Segmentation Pipeline : First a map of the environment is generated using traditional mapping techniques. Using an ICP based SLAM algorithm (PointSlam) and ray-tracing (PointRay), the map is annotated without human supervision. This annotated data is further passed as supervision to a 3D convolution network, KP-Conv [3]. The model is then updated iteratively over consecutive sessions.

Chapter 2

Hardware

2.1 Hardware Specifications

For this project, we have used the Clearpath Jackal UGV robot. Jackal is a small, rugged and easy-to-use mobile robot presented by Clearpath Robotics. It is integrated with ROS-Kinetic for out-of-the-box usage. Jackal includes an onboard CPU, as well as basic IMU and GPS. For this project, the Jackal has also been equipped with an Nvidia Jetson AGX Xavier for higher compute and a Velodyne VLP-32E lidar for perception. As shown in Fig. 2.1, the jackal robot is charged wirelessly using a Wibotics Wireless Charger - TR-301. The TR-301 is typically paired with the higher-powered onboard charger.

Table 2.1: Specifications of the Jackal Robot

Sr.No.	**Specifications**	**Technical Details**
1.	External Dimensions	$508 \times 430 \times 250mm$
2.	Weight	17kg
3.	Maximum Payload	20kg
4.	Maximum Speed	2m/s
5.	Runtime	4hrs
6.	Drivers and API	ROS Melodic, ROS Kinetic, Windows10

The Jackal robot comes with a 270 watt-hour lithium battery pack, Sony Bluetooth controller, and the Jackal User Manual. We will discuss how the robot was set up and some issues faced in setting up.

2.2 Robot Setup

The Jackal Robot comes with a user manual with instructions to set up the robot. The first step is to power up the robot and move it using the provided Sony Bluetooth Controller. One can move the robot by pressing the L1 key and moving the left joystick to operate the robot at a nominal speed. The robot can also be run at turbo (max speed) by pressing the R1 button while moving the left joystick in intended direction. However, it is always recommended to try the turbo mode only in free spaces where the robot may not harm itself or the surrounding. One can also change the parameters for maximum acceleration and speed in the jackal control configuration files if needed.

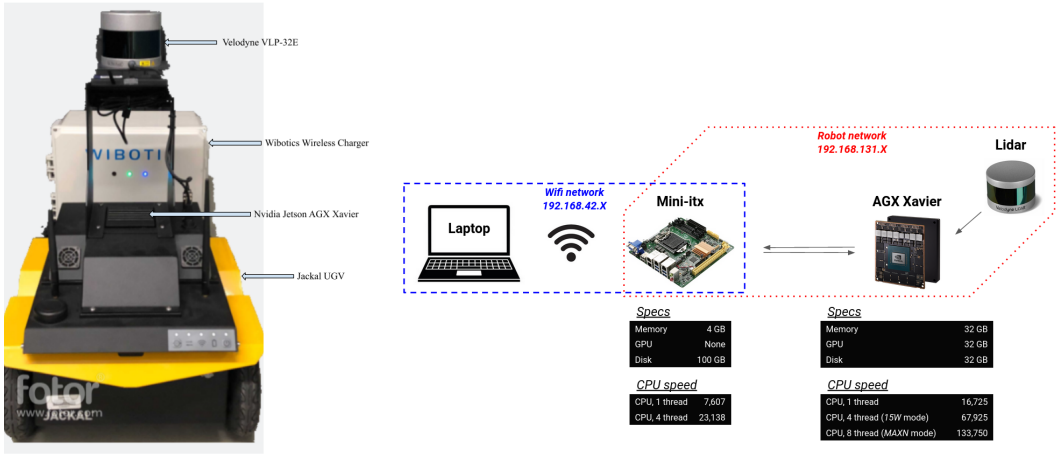

Figure 2.1: Jackal Robot and System Network: The lidar is connected to the AGX Xavier. The Mini-itx computer is the ROS Master, running the jackal bringup launch files to start and keep the robot running. AGX Xavier is connected to the same network with the mini-itx acting as the ROS Master. Finally, we can connect to the mini-itx through local pc.

2.2.1 Possible Issues with the Setup

We faced some issues while setting up the bluetooth controller with the robot and charging issues which are discussed in this section along with the probable solutions that could be tried. This section is formatted in order to provide guidance in case one runs into similar issues.

1. Unable to connect the controller to the robot

   (a) Approach 1 - Ensure the controller is fully charged and check if the bluetooth is enabled by running sudo service bluetooth status in the terminal. If bluetooth is not enabled, start the bluetooth daemon with

   ```
   $ sudo service bluetooth start
   ```

   and make sure the bluetoothd is executable with

   ```
   $ ls -la /usr/sbin/bluetoothd
   ```

   Power cycle the controller before trying to pair it. Finally, press the PS and share button simultaneously so the controller Bluetooth is enabled for pairing and use

   ```
   $ sudo ds4drv-pair
   ```

   (b) Approach 2 - Run the command

   ```
   $ sudo sudo bt-device -l
   ```

   and check the paired devices (Wireless Controller). Note the MAC address of the controller and run

   ```
   $ sudo bt-device -r AA:BB:CC:DD:EE:FF (MAC Address)
   ```

   This will unpair the controller, and remove any existing configuration files for it. We will re-pair the controller using "sudo bluetoothctl".

```
$ sudo bluetoothctl
  scan on
```

Put the controller in paring mode by holding the Share + PS buttons. Wait for the "Wireless Controller" with the appropriate MAC address to scroll by and run

```
scan off
trust AA:BB:CC:DD:EE:FF
pair AA:BB:CC:DD:EE:FF
connect AA:BB:CC:DD:EE:FF
```

Once it's connected, and the light turns blue, exit bluetoothctl by pressing ctrl+d. Check the output of

```
$ ls -l \dev\input
```

to see if there is a pairing between PS4 and the JS0 or JS1 port. Check the outpute of the controller using jstest or jstest-gtk. Irrespective of the pairing shown by the previous command, make sure to check if the controller is giving any output on either the JS0 or JS1 port and change accordingly. If there is still an issue in connecting the controller to the robot, check the antenna cables attached to the integrated computer to see if any cable is loose and try again.

2. Charging Issue

   (a) The Jackal robot is charged wirelessly using the Wibotics Wireless Charger. While charging the robot, make sure the robot is perfectly aligned with the transmitter of the charger and the LED on the charger turns blue indicating that the charging process is turned on. In case the robot gets discharged, the protective circuit on the battery will not allow any charge from the battery even to charge up the onboard charger required to pair with the wireless charger. To check if the onboard charger is being detected by the TR-301, check the paired chargers on the Wibotics Control Panel. If the onboard charger has completely powered off, and the robot is not able to charge through the wireless charger, try powering the robot through a wired charger if available. Otherwise, as mentioned in the TR-301 user manual, check if the battery is a protected battery and if it is safe to resume the charging by providing a small recovery charge. If an unprotected battery is over-discharged, attempting to recover it in this way may lead to permanent battery failure or even battery fire. In case of the discharged on-board charger, the receiver of the charger fails to get detected by the TR-301. In this case, one could provide an external $5V$ to the receiver to enable the connection and start the ERC (Enable Recovery Charge) to the battery. It is highly recommended to read the user manual before recharging a discharged battery or contact the Wibotics Support Center for guidance.

3. Visualizing Lidar frames

   (a) As shown in Fig. 2.1, the Velodyne VLP-32E lidar is connected to the Nvidia Jetson AGX Xavier board. Due to the mini-itx acting as the ROS Master, it is not possible to directly view the velodyne frames on the local computer. A solution to this issue is to republish the topics relevant for visualization from the Xavier board and use these republished topics to visualize the relevant topics on RVIZ on the local computer.

The next step in setting up the robot is networking. This can be easily done by following the video provided by Clearpath Robotics on how to set up the networking of Jackal Robot. As the video is publicly available online with detailed instructions, we will not go over this step.

Finally, the robot is set up to autonomously navigate. To test the setup, we use the basic ros navigation packages of gmapping, TEB (time-elastic band) planner, and the jackal-navigation packages provided by Clearpath Robotics. On fine-tuning the parameters of TEB planner, the robot worked smoothly and ready for incorporating the PointMAP and a modified TEB planner as explained in [1] and [2] respectively.

Chapter 3

Motion Distortion

3.1 Automated Annotation Procedure

We use multi-session SLAM and ray-tracing as annotation tools for the training of the semantic segmentation networks. The automated annotation procedure as explained in [1] consists of two parts

1. PointMAP - An ICP based SLAM algorithm to provide a point-cloud map.
2. PointRAY - A ray-tracing algorithm to calculate the movable probabilities in the map.

The PointMAP algorithm is further divided into two parts - sparse frame to map alignment using ICP and a map update function that constantly updates the map given the current-best normal scores. Given a reading point cloud P and a reference point cloud Q, ICP aims at finding the transformation T that minimizes the alignment error between P and Q given the objective function :

$$E(T) = \sum_{(p,q \in K)} ((p - Tq).n_p)^2 \tag{3.1}$$

, where n_p is the normal of point p.
We use the latest odometry of the robot as the pose to solve the initialization issue of local refinement algorithms like ICP. For efficiency, the map is updated only with the sub-sampled frame points used in ICP. The implementation details are explained throughly in [1].

Although this algorithm works perfectly in simulated environments, we face some issues while implementing it in real world which we will discuss further. One of the major issues that arise due to moving sensors like the lidar is the problem of motion distortion.

3.2 Motion Distortion

Motion distortion is a phenomenon caused due to sweeping-while-moving sensors like a lidar, radar, rolling-shutter camera. The basic working principle of a lidar involves a laser-range finder reflected by a rotating mirror, thus gathering distance measurements at specific angle intervals. For lidars that are able to cover a view of 360°, work on a similar principle with a fan of lasers attached vertically with different angles. Due to these mechanically moving parts of the sensor attached on a moving robot, it is impossible to achieve the position of all the world points at exactly a given timestep. As shown in Fig. 3.1, let us assume the lidar starts spinning at the initial position and is attached on a moving robot. Thus, by the time the lidar has completed one complete rotation, the robot has moved a significant amount. For the generation of the point cloud, it is assumed that all the points in the same scan are measured from the same origin, even if the robot is moving at high speed. Due to this change in origin of the lidar

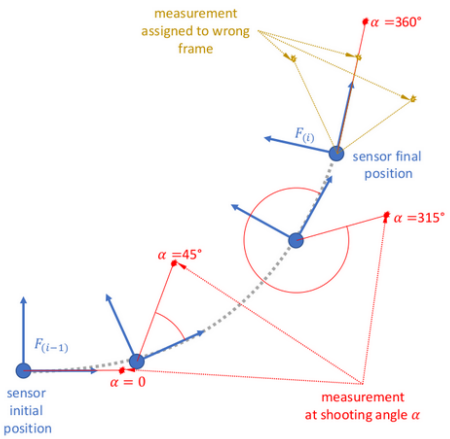

Figure 3.1: Motion Distortion[4] : Consider a lidar attached on a robot is moving while acquiring data. By the time, the lidar completes an entire revolution, it has moved along with the robot. Thus, causing a distortion in the registered point cloud.

while gathering data, the resulting points have different reference locations in space. This results in a distorted point cloud, with points in front of the robot appearing further than they are and the points behind the robot appearing closer than they are. Using algorithms like ICP, we can find the pose of the robot T as the composition of all rigid transformations from the start of the trajectory. However, the error due to motion distortion will accumulate during each iteration of the ICP algorithm, causing a significant degradation in the map. The effect of motion distortion can be predominantly seen when a differential-drive robot starts rotating whilst taking data from the lidar.

Point clouds play a significant role in robotics to provide accurate representation of the environment. Accurate point clouds are important for precise localization and mapping. Such a skewing caused due to motion distortion can lead to localization failures if the skewing is significant. Point cloud registration algorithms like the Iterated Closest Point (ICP) have proven effective in mobile robot localization, but, they fail when the robot is moving at high speeds. As our approach relies on an ICP based solution for point cloud registration, it is limited to operation in slower speeds due to the motion distortion. Hence, it is of utmost importance to incorporate a correction for the motion distortion, to allow a faster operation of the robot without degradation of mapping and localization.

3.2.1 Approach

Motion Distortion Correction is a well studied problem for robots moving at high speeds over uneven terrains [5], [4]. However, for our use-case, we will never face extreme conditions with robots moving at high speeds and accelerations. Hence, we choose to use an easy but robust solution for de-skewing of the point cloud.

The lidar motion is modelled with linear angular and translational velocities. Let us assume that the lidar scan is rotating at a constant rate. In this simple case, we can linearly interpolate between the two poses at different times given a suitable interpolating factor. A number of choices can be made to choose the interpolating factor like angles, point indices (as the indices are proportional to the angles), timestamps of individual points etc. As we have the timestamp of every point from the lidar scan, we can accurately interpolate between the two poses using the timestamps as the interpolating factor. This method based on linear interpolation would not be true in cases where the robot motion is not linear,

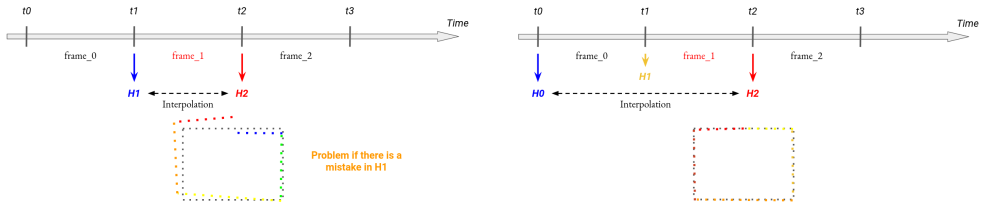

Figure 3.2: Experimental De-Skewing: Theoretically, the de-skewing algorithms should work with the linear interpolation between the two poses at the start and the end of lidar sweep. The left figure shows that if there is an error in the pose at the start of the lidar rotation, it can cause the ICP algorithm to fail. Thus the alignment between the map (in gray) and the frame (in rgb) is distorted. However, as per the right image, interpolating between the previous pose and the current pose gives better results.

for example, a robot is wobbling on an uneven surface causing non-linear oscillations to the lidar while data-collection. However, for the sake of simplicity and the given use-case we can ignore such cases.

Let t be the current timestamp and t_k be the timestamp when the lidar began to sweep. $T^L_{k_0}$ is the lidar pose transform between the timestamps $[t_k, t]$. Given a point i at timestamp t_i, let T^L_{ki} be the transform between $[t_k, t_i]$.

$$T^L_{ki} = \frac{t_i - t_k}{t - t_k} T^L_{k_0} \tag{3.2}$$

3.2.2 Experiments

The experiments were conducted on the Jackal UGV robot equipped with a Velodyne VLP-32 lidar. To test the robustness of the de-skewing algorithm, tests were conducted on a robot rotating in-place at angular velocity of 3 rad/s. The effect of motion distortion is dominant in such conditions of fast rotations. Due to the robustness of ICP algorithm and a robot travelling at moderate speeds, such a technique of linear interpolation between poses should work well in theory. However, there are certain aspects related to the inaccuracies in odometry readings and computation time, which may lead to failure in the de-skewing caused due to motion distortion.

During the experiments we found that the motion distortion was still prevalent after the incorporation of the described correction in the PointMap algorithm.

3.2.3 Results

Through the experiments, the major issue for the failure of the de-skewing algorithm was found to be due to the inaccuracies in the initial transforms provided to ICP. ICP is a local refinement algorithm and depends greatly on a good initial estimate. In the absence of a good initial estimate, the ICP fails to converge, thus leading to improper alignment and degradation in the quality of the map. As this map is used in further iterations too, the error starts to accumulate leading to a distorted point cloud.

Given the previously explained method for de-skewing, we linearly interpolate between the pose during the start of the sweep and the current pose at the end of the lidar rotation to get a correction for the drift due to motion distortion. As shown in the toy-example in Fig. 3.2, if there is an error in the current pose at the start of the sweep, the ICP gets a bad initialization, thus causing the ICP solution to fail. This error may be caused due to noise, non-linear trajectory, etc. However, we can solve this issue by interpolating between the previous known pose and the current pose and apply this correction to

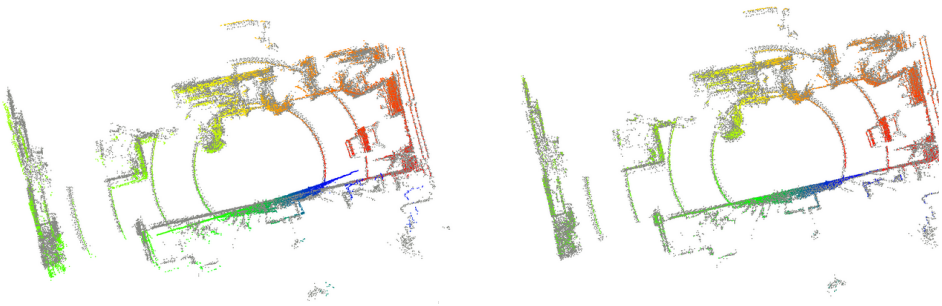

Figure 3.3: Qualitative Results: The images show the alignment of lidar frame(rgb) on the map(gray) before and after applying the correction for motion distortion. The left image shows the distortion caused in the alignment due to motion distortion. The right image displays the proper alignment between the frame and the map when the robot is rotating.

every point at the start of the ICP loop. As we can assume that the previous known pose is proper due to the applied corrections in previous timesteps, the interpolated solution proves to be a good initialization to the ICP algorithm. Fig. 3.3 shows the qualitative results for the correction due to motion distortion with the robot rotating along a fixed axis.

Chapter 4

Initial Alignment in Multi-Sessions

Consider the case of annotation across multiple sessions. An important issue to take into account is the session alignment. As mentioned in [1], PointRay requires the frames to be perfectly aligned with the map we want to annotate. The current system assumes the robot starts at the same initial position in every session, thus providing proper initial alignment after applying ICP. However, it would be better to have a good initial alignment irrespective of the starting point of the robot, thus removing the constraint of starting the robot from same point in every run. This problem could be solved through a global registration pipeline followed by a local refinement using ICP.

4.1 Global Registration

Registration of 3D surfaces is a fundamental problem in computer vision. The problem of point cloud registration becomes even more difficult when the surfaces are partially overlapping or the initial estimate is not given. The problem of registration has been divided into two parts namely global methods and local methods.

According to [6], local refinement methods like the ICP require a good initial guess for them to work. Local refinement algorithms like ICP, begin with a rough initial alignment and alternate between establishing correspondences via closest-point lookups and recomputing the alignment based on current correspondences. However, ICP works only when we have a good initial guess, and is unreliable when the initial estimate is not accurate enough. Generally, global registration methods based on Ransac provide a loosely-fit initial estimate to the ICP algorithm which is used to refine the estimate locally. However, these two step methods of ransac based global registration followed by ICP based local refinement is computationally expensive and takes too much time.

In their work [7], the authors suggest a fast global registration algorithm for partially overlapping 3D surfaces. They claim that the algorithm is an order of magnitude faster than global methods (even faster than local methods like ICP) and exceeds the accuracy of the state-of-the-art global registration methods. [7] presents a fast global registration algorithm which does not include any iterative sampling, model fitting, local refinement or any initial alignment. Thus saving a lot of computation on the iterations. It optimizes a robust objective defined on dense surfaces.

$$E(T) = \sum_{(p,q \in K)} \rho ||p - T(q)|| \tag{4.1}$$

, where p and q are the point features of point clouds P and Q, obtained by fast point feature histogram method (FPFH) and ρ is a robust estimator. Due to this dense coverage, the algorithm claims to produce an alignment that is as precise as that computed by well-initialized local refinement algorithms.

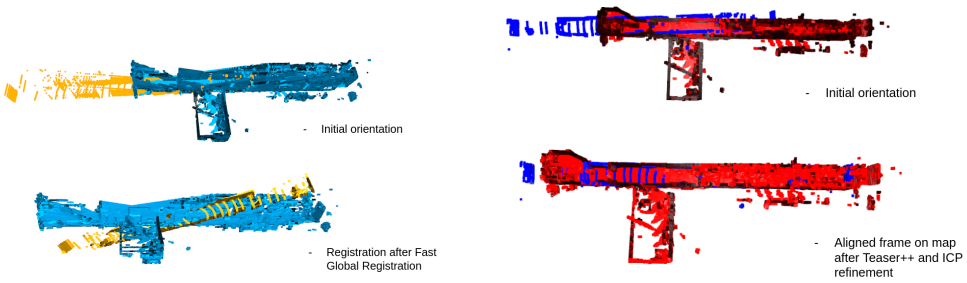

Figure 4.1: Global Registration: The left images show the result of pointcloud registration of a frame(yellow) and the map(blue) using the fast-global registration method. The right image shows the pointcloud registration of a frame(blue) and the map(red) using the Teaser++ registration method.

[8] provides a fast and certifiable point cloud registration algorithm. The algorithms tries to optimize the problem of registration using the features of two pointclouds by the following objective function:

$$E(R,t) = \min_{s>0, R\in SO(3), t\in R^3} \sum_{i=1}^{N} min(\frac{1}{\beta_i^2}||b - sRa_i - t||^2, \bar{c}^2) \tag{4.2}$$

, where β is the given bound to truncate the effect of outliers, c is a constant, given two point clouds $A = \{a_i\}_{i=1}^{N}$ and $B = \{b_i\}_{i=1}^{N}$.

We have tried to implement the two described approaches to solve the issue of global registration given partially aligned pointclouds without any initial guess. As we can see in Fig. 4.1, given two arbitrary pointclouds of a sparse lidar frame and a dense map of the same region, the described methods do not seem to do a good job. These global registration methods rely on the Fast-Point Feature Histogram features [9] for calculating the descriptors around the points. However, given the unusual difference in pointcloud densities between the frame and the map pointclouds, such feature descriptors calculated using the neighborhood information do not work very well. The problem with sparse-dense registration is that there is no direct correspondence between each point in two pointclouds but a point equivalent to a set of points. Sparse to dense point-cloud registration has not been explored much. [10] proposes to solve the issue of sparse-dense registration using a clustering approach, however, these methods work for local refinement algorithms and not for global registration which is required in our case.

Some possible solutions for future work could be to explore the option of generating descriptors not relying on the distance between points. FPFH features rely on distances between its neighbors to calculate the weights given to each neighboring point. Due to this dependence on the distance between points, FPFH features may not be suitable for our case of sparse-dense registration. On the other hand, PFH features rely on the angles between the computed normals for the descriptor generation. However, the PFH algorithm is significantly slower compared to the FPFH due to its dependence on all points in the neighbourhood.

Another easy fix could be to manually provide the initial guess to the ICP algorithm using the map generated in the first run. This can be easily done using the 2D Pose Estimate button on RVIZ.

Conclusion

In this report we explained the issues faced in setting up a robot and incorporating systems from simulation to actual hardware. We showed some practical solutions to the hardware issues faced during operation. We identify the problem of motion distortion in the SLAM pipeline and explain an easy but robust solution to solve the issue. Finally, we show a possible constraint in the present system and discuss possible approaches to solve the issue of sparse-dense global registration issue. The future work would include running the entire system in real-world and observe the behaviour of the system trained and implemented in dynamic environments.

Acknowledgement

The author would like to thank Prof. Tim Barfoot and Dr. Hugues THOMAS for their constant guidance and support throughout the project. I would also like to thank the Autonomous Space Robotics Laboratory and Apple Inc. for the access to the robot.

Bibliography

[1] Hugues Thomas et al. "Self-Supervised Learning of Lidar Segmentation for Autonomous Indoor Navigation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 14047–14053. DOI: 10.1109/ICRA48506.2021.9561701.

[2] Hugues Thomas et al. *Learning Spatiotemporal Occupancy Grid Maps for Lifelong Navigation in Dynamic Scenes*. 2021. arXiv: 2108.10585 [cs.RO].

[3] Hugues Thomas et al. "KPConv: Flexible and Deformable Convolution for Point Clouds". In: *Proceedings of the IEEE International Conference on Computer Vision* (2019).

[4] Tobias Renzler et al. "Increased Accuracy For Fast Moving LiDARS: Correction of Distorted Point Clouds". In: *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2020, pp. 1–6. DOI: 10.1109/I2MTC43012.2020.9128372.

[5] Simon-Pierre Deschênes et al. "Lidar Scan Registration Robust to Extreme Motions". In: *2021 18th Conference on Robots and Vision (CRV)*. 2021, pp. 17–24. DOI: 10.1109/CRV52889.2021.00014.

[6] F. Pomerleau et al. "Comparing ICP variants on real-world data sets". In: *Autonomous Robots* 34 (2013), pp. 133–148.

[7] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Fast Global Registration". In: vol. 9906. Oct. 2016. ISBN: 978-3-319-46474-9. DOI: 10.1007/978-3-319-46475-6_47.

[8] Heng Yang, Jingnan Shi, and Luca Carlone. "TEASER: Fast and Certifiable Point Cloud Registration". In: *IEEE Transactions on Robotics* 37.2 (2021), pp. 314–333. DOI: 10.1109/TRO.2020.3033695.

[9] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast Point Feature Histograms (FPFH) for 3D registration". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[10] M. Lamine Tazir et al. "CICP: Cluster Iterative Closest Point for sparse–dense point cloud registration". In: *Robotics and Autonomous Systems* 108 (2018), pp. 66–86. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2018.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0921889017307005.