Automated Robot System Design For 3D Scene Graph Generation

Kang Seong Ju

Department of Physics & Astronomy

Seoul National University

June 26, 2020

1. **INTRODUCTION**

**3D scene graph**

A scene graph is a general data structure used by vector-based graphics or modern computer games etc. It contains information about the logical and spatial representation of a graphical scene. For example, a scene graph might contain graphical data of objects, semantics on objects such as class, material, and relationships with entities.

3D scene graph uses 3D data structure for the graphical data. There have been several 3D scene graph dataset available like in [1].

By far, not many are real indoor-scene point cloud dataset, which can be used for indoor applications or related researches. A real indoor dataset can be noisy and inaccurate but contains information about frequently met situations in daily lives. For that reason, it is important that we capture the real dataset to use in real-world applications.

**Point Cloud**

There are 3 popular ways to represent 3D data, Point cloud, Voxel, Polygonal mesh. To put it simply, Point cloud is a set of points. Each point can have x, y, z coordinates, RGB colors, possibly normal vectors. It can be obtained with RGB-D camera like realsense cameras or 3D sensors like LIDAR. Since Point cloud is used widely as raw data obtained from gadgets, there have been many attempts to apply them in diverse situations.

**ROS**

ROS(Robot Operating System) is widely used open source framework to control a robot or multi-module systems. ROS is a distributed framework of processes called “Nodes”, that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed.

In our work, we use a new version ROS2 for we need a fast data transition between the real-sense camera and the notebook.

***Topic***

Topic is a data bus for nodes to exchange messages between themselves. A node may “publish” data to any number of topics and “subscribe” to any number of topics at the same time.

In this work, we used “amcl\_pose” topic to monitor the robot position.

***Action***

Actions are one of the communication types of ROS2 intended for long running tasks. It consists of 3 parts: a goal, a result, and a feedback.

Actions are preemptable. It also provides steady feedback so that we can monitor the process of implementing a goal.

**Navigation Stack**

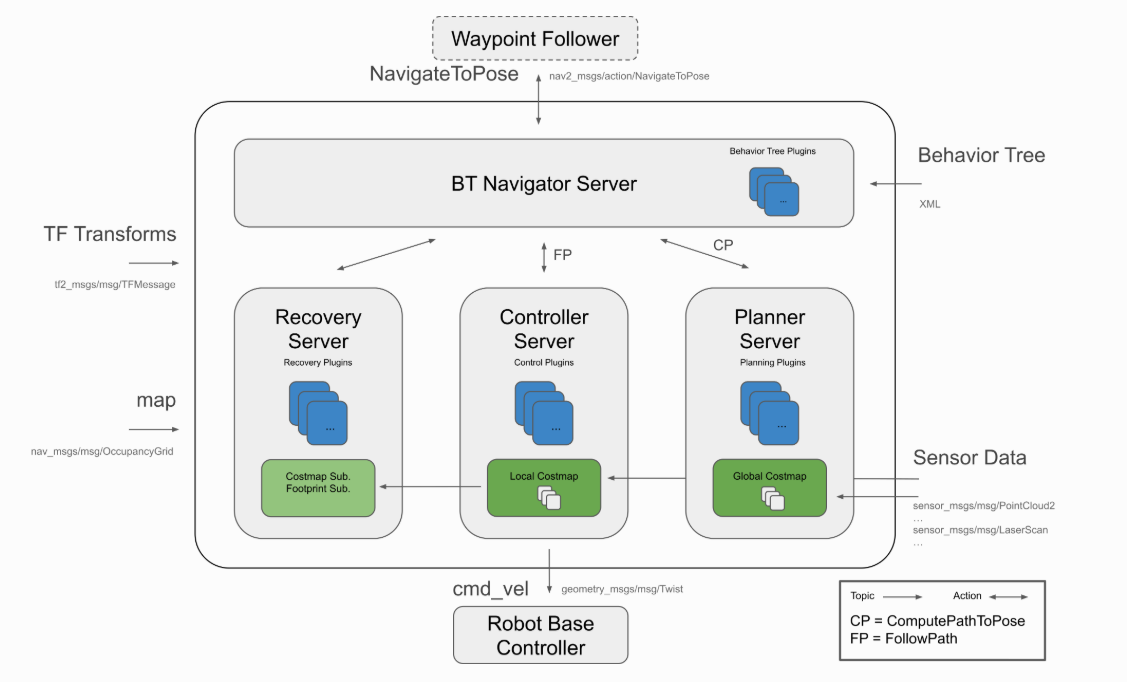


Figure 1 Navigation2

Navigation2 has tools to

1. Load, serve, store maps
2. Localize the robot on the map(amcl[[1]](#footnote-1))
3. Plan a path from A to B(Nav2 Planner)
4. Control the robot as it follows the path(Nav2 controller)
5. Build complicated robot behaviors using behavior trees(Nav2 Behavior Trees and BT navigator)
6. Compute recovery behaviors in case of failure(Nav2 Recoveries)
7. Follow sequential waypoints(Nav2 Waypoint Follower)

All we need to navigate is then giving an action type “NavigateToPose”. Then Navigation2 stack will plan a path, control the robot, build the behavior trees etc.

**Turtlebot3**

Turtlebot series is a ROS based mobile robot which is cheap and widely used for educational, research purposes. It provides various open source softwares for users which can be brought and fixed according to their needs.

In our work, Turtlebot3 model is used to move around in an indoor environment and capture 3D data. Since the mobile robot needs to process large dataset in real time, we mounted an intel core i7 notebook. It gets frames from camera and works as ROS server.

**Detectron2**

Detectron2 is an object detection trainig/inference platform developed by FAIR(Facebook Artificial Intelligence Research). When using Detectron2, one doesn’t have to implement training loops so that one can concentrate on developing model itself. In short, the only thing to do is to define the input and output and the rest is on Detectron2. Also, Detectron2 is known to be faster than other open sources with good python optimizations. Computationally expensive parts are implemented with CUDA and C for better performances.

In this work, we use Detectron2 as our object detection model to detect target objects. The input is color frame which is sent from realsense camera and the output is the detected class and pixel coordinates of objects.

**Registration**

The Captured 3D point represents a parted object viewed from a specific position. Since we need the geometry of objects, we need to merge the captured 3D data and construct the complete geometry. This merging process is called “Registration”.

There are largely 2 categories of registrations. Global registration means roughly matching two data set which are relatively far. ICP registrations do the similar thing on relatively close data. Three kinds of ICP algorithms are used in this work.

***Point-to-point ICP***

Correspondence set K is defined as

p from target point cloud P, q from source point cloud Q. The objective function E(T) is defined by

This algorithm iteratively update the transformation T by minimizing the objective function E(T). Computation includes the residuals and jacobian matrices of the objective function.[[2]](#footnote-2)

***Point-to-plane ICP***

This algorithm is almost the same to Point-to-point ICP. The difference is that it uses a different objective function.

The difference is that it uses the normal vector of point p. Rusinkiewicz[5] showed that point-to-plane ICP algorithm has a faster convergence speed than the point-to-point ICP algorithm.

***HMRF ICP***

Let B be the free point cloud and C be the fixed point cloud.

HMRF(Hidden Markov Random Field) ICP[?] uses hidden field variable Z. corresponds to each and have either +1, -1[[3]](#footnote-3). If >0, it predicts the corresponding couple to be inlier, which means the pair is a right match. If <0, it predicts the corresponding couple to be outlier, which means the pair doesn’t match.

The HMRF algorithm makes use of EM algorithm to find the optimization condition.

E step : It finds the expectation value of . To calculate the expectation value, we need to find the probability of being +1, -1.

*: iterate neighbors near i*

*: a parameter controlling interaction strength.*

*w : edge weight*

*: inlier distribution(normal dist) mean*

*: inlier distribution standard error*

*: outlier distribution(logistic dist) mean*

*s : outlier distribution scale*

*: set of parameters(*

The normalization factor is the same for the two calculations, so is simply their sum. If the neighbors are close to +1, goes high by the exponential term. It goes the same way when neighbors are close to -1. If the likelihood of being a sample from normal distribution is high,

goes higher since HMRF algorithm assumes inliers to follow normal distribution as a prior. If the likelihood is high for logistic distribution, goes high.

In short, it considers neighbors and distribution compositively to determinate the next expectation value of ..

The expectation value is calculated by

M step : The optimization is to maximize the data likelihood defined as

This step finds the optimizing parameters ) based on calculated in E step. The MLEs of the normal and logistic distribution parameters are calculated in the normal way.

For the outliers,

HMRF method iterates this E, M step for sufficiently large steps to find the optimized state. In each iteration, the Transformation matrix is calculated to localize point cloud B and C.

1. **DESIGN**

The overall system design is as below.

1. System gets global path points
2. Navigate to global path points
3. Detect objects
4. Extract local path points
5. Navigate to local path points
6. Capture point cloud

* Go to 5 until end of local path

1. Registration

* Save registered 3D data
* Go to 2 until end of global path

Figure 2 explains the system well. It gets the 2D slam map to know the spatial information about indoor environment. After registration, it returns the location, category, 3D data of objects.

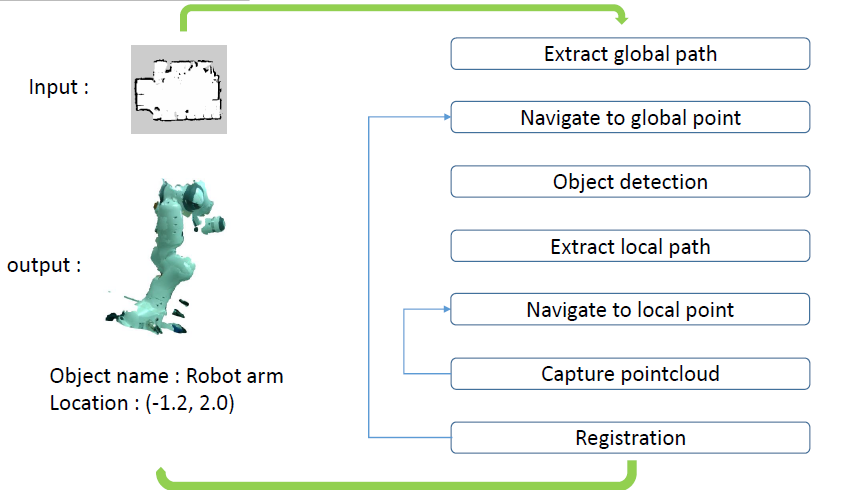
****

Figure 2 overall system

**2.1. Global Path**

The global path point should be away from obstacles enough to avoid collision. As we can read the slam map file and get it as a numpy array, we may know the location of obstacles. Then it is possible that we get safe global path which is collision-free. The overall algorithm is below.

**Global Path Algorithm**

|  |
| --- |
| Load map  r : radius from obstacles  for each obstacle pixel :  draw a circle of radius r  for each circle point :  if color is not black :  colorize yellow  for each inside-circle point :  colorize black  uniformly sample from yellow points |

In this way, we can get the right global path(yellow points) which is away enough from obstacles. Then we uniformly sample points from this path to get global path points. When Turtlebot gets to global path point, it will start object detection.

**2.2. Object detection**

Object detection is done by Detectron2 and realsense camera d435i. We can get depth frame and color frame from realsense camera. These frames are then put into Detectron2 to get the bounding box of objects.

The center coordinate of bounding box and depth frame is used to get camera coordinates using camera intrinsic matrix. Since we know the odometry of robot and camera extrinsic, finally we get global coordinates of the detected object.

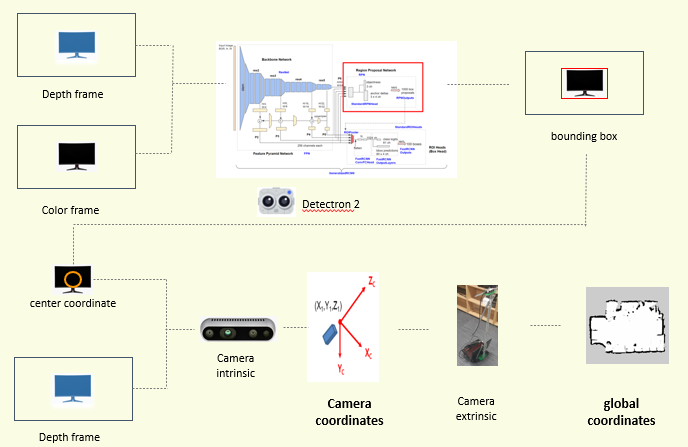
****

Figure 3 Object detection

**2.3. Local Path**

After object detection, Turtlebot needs to move around the object to capture 3D data from different view. We need to consider if it is safe from obstacle collisions, and if the points are close enough to make registration possible. Since global path is safe from collision, it can be reused as local path points. We draw a circle from the initial robot point and meeting points between global path and the circle are set as local path points.

We put ‘local radius’ as the system input so that we can tune the parameter.

Local Path Algorithm

|  |
| --- |
| Draw a circle from starting point  Local\_path = [ ]  For each circle point :  If yellow :  Add to local\_path  Return local\_path |

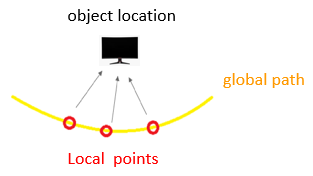


figure 4 Local path points

**2.4. Capturing 3D point cloud**

At each local path points, the system is paused for a while and captures 3D point cloud. Each point cloud is saved with its location information so that we can use it for registration.

**2.5. Registration**

In Open3D, there are 2 kinds of ICP registartions implemented as library functions, Point-to-point and Point-to-plane. We want to explore the difference in performance between them. Furthermore, we investigated HMRF ICP altogether to compare between ICP registration algorithms.

**3. EXPERIMENT**

**3.1. Global path**



R : 1 m

Global path points number : 20

Since we can capture 3D data from 1m away by using realsense API, R = 1m would be good enough.

**3.2. Object detection**

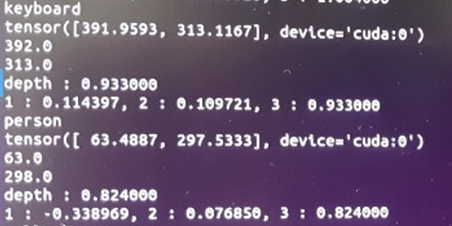


Figure 5 Detected objects

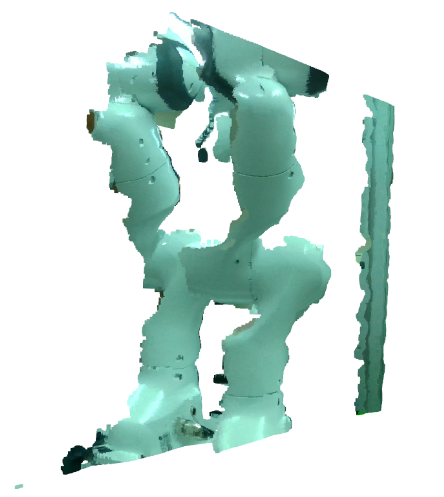
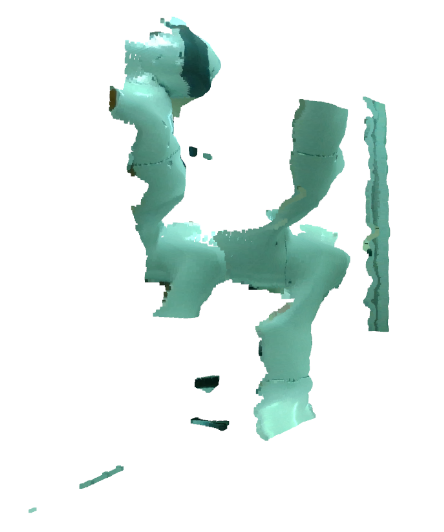
Figure 5 is the detection log. In this log, Keyboard and person is detected and their pixel coordinates are printed. Since we need to know the camera coordinates, we used camera intrinsic matrix to get the camera coordinates. The ‘1’, ‘2’, ‘3’ corresponds to x, y, z axis of camera coordinates.

**3.3. Registration**

For registration, we captured two 3D point cloud of robot arm(FRANKA PANDA) from slightly different views.

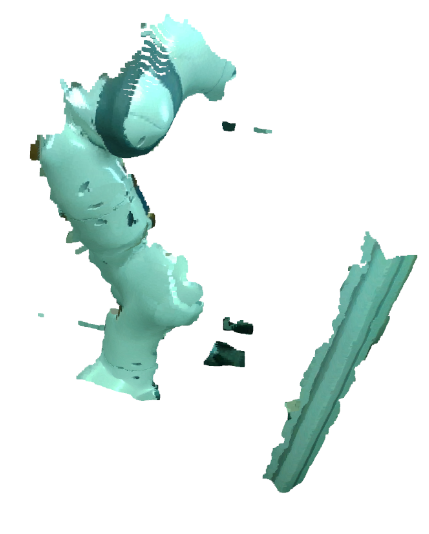
The 3D point clouds are cut in the vertical axis for some proportion. The overlap proportion is calculated by (matching pairs number)/(entire pairs number) of the registred data.

We take different proportions and different algorithms see if ICP registration process succeeds for each case.

2.Cut for some proportion.

1.Raw data.

3.Global

registration.

4. ICP

registration.

**Point-to-point ICP registration**

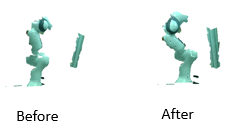
****

Figure 6 Point-to-point ICP

The overlap proportion is **4.37%**. As you can see, the registration succeeds at this low overlap.

**Point-to-plane ICP registration**

****

figure 7 Point-to-plane ICP

The overlap proportion is **7.09%**. This proportion is the limit. After this proportion, we could see that registration fails.

**HMRF ICP registration**

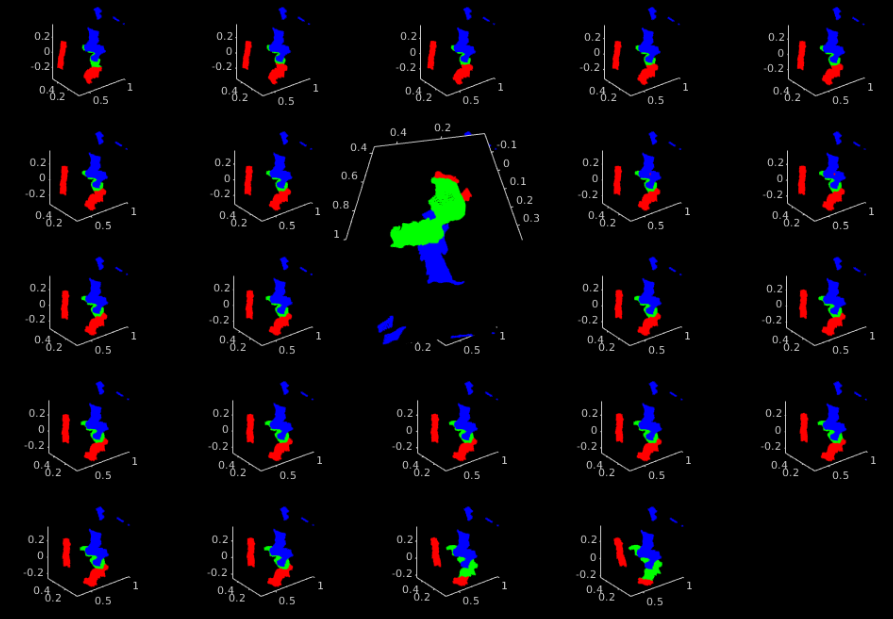
****

figure 8 HMRF ICP

The overlap proportion is 10.45%. Blue represents reference point cloud, red and green represent outliers and inliers of source point cloud respectively. Figure 8 shows each steps of iterations and the last configuration is moved to the center.

As you can see, source is totally mismatched and the inliers dominate in the end. In processing steps, we can see that green points are gradually increasing. Actually, matching is almost done by global registration from the first. The green should decrease and red should increase a little for a right match since ICP registration is just for minute adjustments.

It is important that we discriminate the inliers and outliers through the right guess. HMRF algorithm seems to take outliers into inliers when the outlier portion is high. This makes chain action such that inliers increase, the neighborhood is affected by these inliers, and again inliers increase.

The reason is maybe the assumed outlier distribution, *logistic distribution*. As the inlier points are close to their pairs, we can easily guess that inliers follow normal distribution which has small mean value. However, there is no guarantee that outlier should follow logistic distribution. Alternatively, normal distribution is used[[4]](#footnote-4) for outliers, but this is not plausible since outliers are totally irregular.

Therefore, it is hard to bring the right prior for outlier distribution ending up with wrong guess.

**3.4. Registered 3D data**

For the last, we combined the whole modules and checked if the right 3D data is produced.

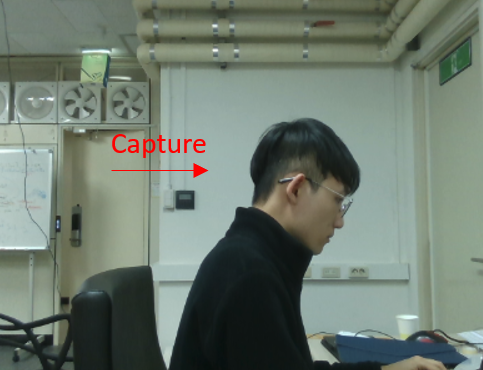


figure 9 Capturing person

**Processed data**

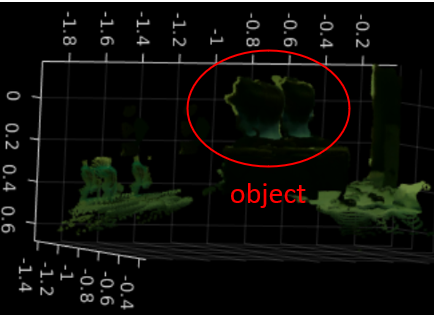
****

figure 10 Object(person)

Figure 10 is the result of 3 point cloud registered.

There seems to be two point cloud in the final data which means registration went wrong.

This maybe because that the camera extrinsic is not correct. Since we measured the camera extrinsic with tape measure, the extrinsic is not so credible. As ICP registration works when the global registration is quite accurate, it is crucial that the extrinsic is accurate.

Also, Turtlebot odometry is not credible. Turtlebot control is moving quite fast, but we couldn’t make sure that the published odometry information is frequent enough. As Turtlebot is a cheap robot and the open source is not so accurate, it might be the cause of inaccuracy.

**4. CONCLUSIONS**

3 kinds of registrations are compared. Among them, point-to-point registration showed the most robust performance to low overlap. HMRF showed low performance despite its high expense in calculation. It seems that using point-to-point ICP or point-to-plane ICP would be good enough.

The registered data from the system workflow showed bad quality. We should design a method to measure the camera extrinsic more accurately. Also, Turtlebot control might be the problem. Since we bring the raw package, the control algorithm may not good enough. Turtlebot moves and stops abruptly which might cause slight inaccuracy in odometry information. More efforts should be made on revising the control source code to improve the overall performance of our system.

**REFERENCES**

* [1] Y.Chen and G. G. Medioni, “Object modeling by registration of multiple range images”, Image and Vision Computing, 10(3), 1992.
* [2] John et al, “Robust low-overlap 3D point cloud registration for outlier rejection”, International Conference on Robotics and Automation, 2019
* [3] Paul J. Besl and Neil D. Mckay, “A Method for Registration of 3D Shapes”, PAMI, 1992.
* [4] Armeni, I et al, “3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera”, In Proceedings of the IEEE International Conference on Computer Vision, 2019.
* [5] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm”, [Proceedings Third International Conference on 3-D Digital Imaging and Modeling](https://ieeexplore.ieee.org/xpl/conhome/7363/proceeding), 2001.

**Python code :** <https://github.com/djflstkddk/globalpath>

1. Amcl pose contains the robot odometry information. [↑](#footnote-ref-1)
2. [1] [↑](#footnote-ref-2)
3. This is an analogy of ‘spin’ from physics. It is similar with Ising model in that Ising model uses neighboring spins to predict the next state spin configuration. [↑](#footnote-ref-3)
4. [2], 4p. [↑](#footnote-ref-4)