Mobile Lost and Found

Li-Wei Yang
Department of Biomechatronics Engineering
National Taiwan University
Taipei, Taiwan
b07611001@ntu.edu.tw

Hua-Ta Liang
Department of Biomechatronics Engineering
National Taiwan University
Taipei, Taiwan
b07611038@ntu.edu.tw

Andrew Jian An Tay
Department of Biomechatronics Engineering
National Taiwan University
Taipei, Taiwan
b07611024@ntu.edu.tw

Cheng-Yen Chung
Department of Biomechatronics Engineering
National Taiwan University
Taipei, Taiwan
b07611048@ntu.edu.tw

***Abstract*— Lost properties among campus has become an annoying problem: we often see people posting pictures of lost properties in online group to search for them. In this research paper, we proposed a mobile platform to help people finding their lost properties in a confined space using RGB-D camera and lidar. The system comes with an intuitive GUI and a command prompt for user to interact with.**

Keywords— ROS, SLAM, BRISK, Mobile robots, RGB-D camera

I. Introduction

"Searching" has been one of the most common problems we face in daily life, statistics show that people spend an average of 5,000 hours on searching things at home every week. So far we have not seen any commercial product that would assist users to find their belonging autonomously. If such a repetitive problem could be solved by robots, we think it would generally improve human life style.

The Original plan was to use RTAB-map [1] to do visual SLAM and if possible, do object detection on the map first before having the robot moving out onto the field to do object detection with CNN such as YOLOv4 [2]. Methods such as A-CNN [3] already exist to deal with 3D point cloud detection, improvement of such a Convolution Neural Network is only a matter of time, doing object detection on cloud map is possible. Another possible method was the map image fusing and patching method [4] research by fellow students of National Taiwan University. With patched 3D maps, we could do image detection directly on the map with 2D CNN, just as how it looks from a camera. Either way, conducting a pre-screening search is viable before conducting a brute force search on the field directly.

Due to the incapability of our hardware, we are only capable of running Rtabmap on our Desktop computer but not on our companion computer (Jetson Nano) on board, the visual odometry would lose track of itself due to computing latency, the solution is simply, a better on board computer.

Because of the situation above, we replace the most ideal structure above with Gmapping for 2D SLAM and pure image processing (BRISK algorithm [5]) for object detection.

II. Related Work

For object detection using template matching, SURF (Speeded Up Robust Features) [6] and SIFT (Sorting Intolerant From Tolerant) [7] are two most prominent methods, but they are patent protected and thus our research does not use them. Another option proposed by Ethan Rublee et al. is ORB (Oriented FAST and Rotated BRIEF) [8], which is faster than SIFT. However, due to the constraint of our companion computer, we need a method that occupy less resource, so the resource can be used for SLAM. Thus, we choose BRISK [5] algorithm, which is faster and more efficient.

For Visual-SLAM algorithm, RTAB-map provides better presentation, and does not require a Lidar to works. The computational burden, however, is a huge drawback of this method. For Lidar based SLAM algorithm, hector SLAM [11] seems to be the best option for us: it does not require odometry data and it meets our final goal: it is stable even in a rugged terrain. The high requirement of Lidar precision, however, does not meet our hardware specification. As a result, we use Gmapping as our algorithm—although it only works in flat surface, the map building speed can be faster compared to hector SLAM.

For object searching robots, Kristoffer Sj¨o et al. proposed a mobile robot equipped with monocular camera using SIFT algorithm [14]. But the paper does not depict the GUI of their robot, and the depth data of the object is not collected. Carlos Astua et al. compared contours and descriptor techniques, and combined them to come up with a new technique [13]. The method is unique but again the robot lacks user interface and the coordinate of the objects do not be recorded. Vui Ann Shim et al. proposed a robot platform with only RGB-D camera using SURF to do the task, but since the robot only rely on camera to navigate, the robot cannot avoid obstacles approaching from the back. As a result, we proposed this research to fill the gap of the GUI and the localization of the object.

III. Methods

The below sections are separated as follows: First we would list the specification of the robot, to make sure the reproducibility of our research. Then, we would state our method for object detection as well as navigation. Lastly, we would talk about the transform relationship between the robot and the environment. Two additional sections are about the methods we tried but did not implement: RTAB-map and YOLOv4.

A. Hardware Specification

We use Jetson Nano B01 from Nvidia as our companion computer, with ROS Melodic running as master. The remote computer is equipped with ROS Noetic. The map and the TF are visualized by Rviz on the remote computer, as well as Find Object 2d GUI. We use RPLIDAR A1 from Slamtec, for

Gmapping algorithm, RealSense from Intel as the RGB-D solution, for Object detection. Our Motor driver is Stm32f103rct6, and our motors are equipped with encoders, together with GY-85 IMU to provide odometry data.

B. *Find Object 2d*

Find object 2d is an integrated ROS package with various feature detectors and descriptors to choose from. In this research, we use BRISK (Binary Robust Invariant Scalable Keypoints) algorithm as both detector and descriptor, because it computes faster and less of a computational burden to Jetson Nano than SIFT or SURF. For decriptor matching method, we use bruteforce-hamming. For homography, we use RANSAC (RANdom SAmple Consensus) as our homography method and to exclude feature points outliers. The whole process flow is as figure 1.

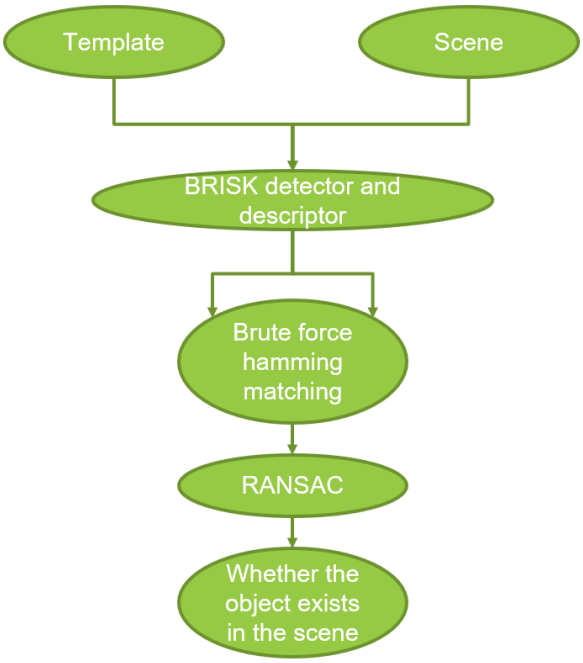

Fig. 1. Object detectoin flow.

C. *TF Tree*

Our tf in Rviz is as figure 2. We specially measured the orientation that the camera relative to the base_link to match the actual bending configuration (x y z in m and yaw pitch roll in rad = 0.06 0 0.35 -1.5708 0.349 0). Figure 3. shows the TF tree of our system. Odometry plays an important role in the system.

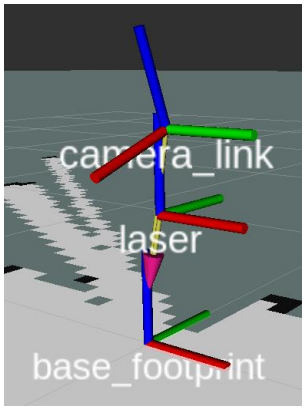

Fig. 2. TF frames in Rviz.

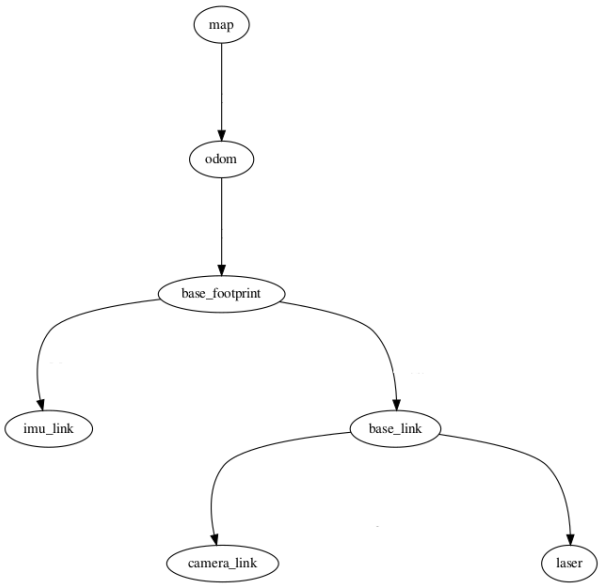

Fig. 3. TF tree of our system

D. *Navigation*

- SLAM system framework (See Fig. 4)

In a SLAM system, Gmapping obtains lidar information and odometer data to dynamically generate 2D grid maps [9]. The navigation package uses grid map, odometer data and lidar data to make suitable path planning and positioning, and finally converts it into the speed command of the robot. The following figure shows the framework of the SLAM system.

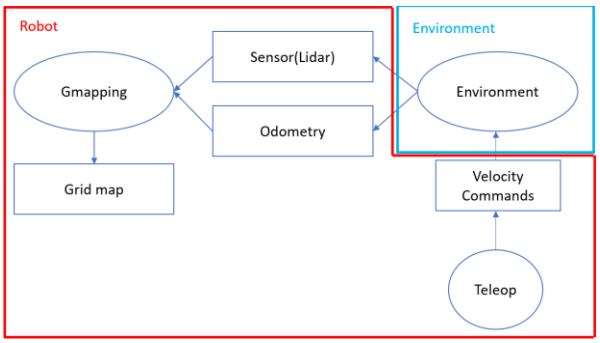

Fig. 4. Framework of the SLAM system

- Navigation system Framework (See Fig. 5)

"move_base" module is responsible for scheduling navigation behavior, including initializing costmap and planner, monitoring navigation state and changing the navigation strategy in time.

As for behavior control, several recovery_behaviors are defined to specify the behavior of the robot after a problem occurs during the navigation process. The recovery steps are as follows:

1. move_base first initializes the global_planner and the local_planner modules to generates their own cost map (global_costmap and local_costmap) through costmap components.

2. By global path planning, the global path of the robot to the target position is calculated, and it is realized by finding the best route based on the cost search of the grid map.

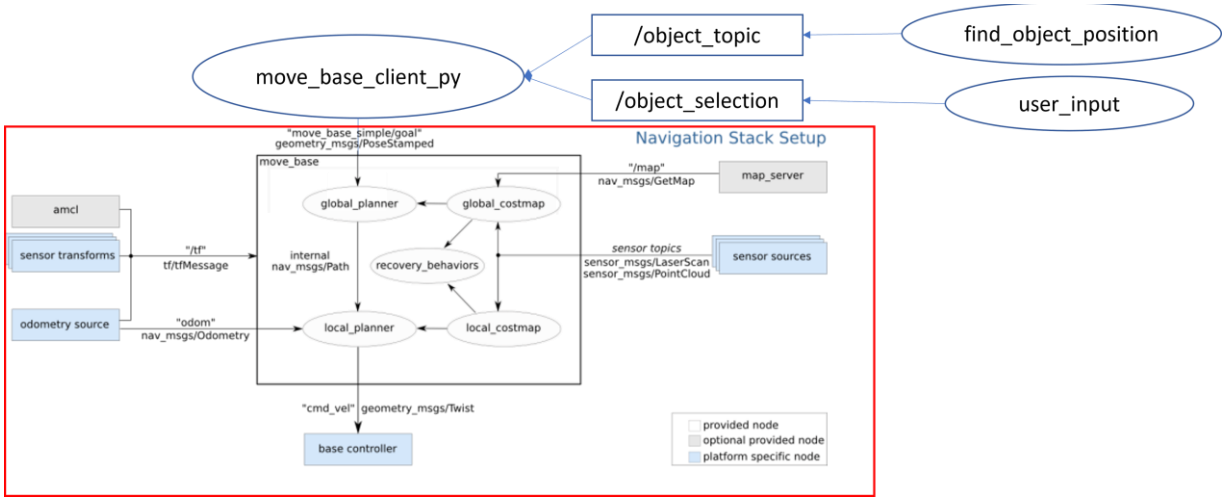

Fig. 5. Navigation system Framework and its integration with Mobile Lost and Found topics.

3. By local planning, it is responsible for the planning of local obstacle avoidance. It will be realized by DWA (Dynamic Windows Approach) [10] algorithm.

- Brief steps of DWA

1. From the kinematic model of the moving chassis, we can obtain the sampling interval of velocity.

2. In the sampling interval, compute the objective function for each sample.

3. Get the desired velocity, interpolate into the trajectory output.

4. During operation, make planning (wake up the planner), operate (calculate the legal speed and publish), clean up (recovery_behavior) and other operations according to the state of the robot.

- Integration to Lost and Found robot (See Fig. 5)

1. To implement the navigation system into our lost and found robot, we first create a /find_object_position node to send lost properties positions and subscribe the information by /move_base_client_py node.

2. As for navigation, we create a /user_input node for the user to type desired lost object ID and sent it to /move_base_client_py node.

3. By the previous information we've got, we can easily find the corresponding position we want to achieve. Later, we send the specific position to the navigation by move_base and it should lead us to the location by the navigation algorithm mentioned above.

E. RTAB-Map

RTAB-Map or Real-Time Appearance-Based Mapping is a SLAM algorithm which support RGB-D camera, Stereo camera and Lidar-based SLAM. Our original idea was to use RTAB-Map for mapping and navigation with the RealSense camera mounted on the robot. However, due to Jetson Nano's limited computing power, we are not able to obtain a satisfying map result. The main problem is that RTAB-Map require massive amount of computing power in order to process feature point on the RGB image to accurately identify the odometry of the robot. (visual odometry method)

Figure 6 is the mapping result of our RTAB-Map build with a laptop. Notice that it is more accurate and satisfy the requirement to be usable. Thus we think it is actually feasible

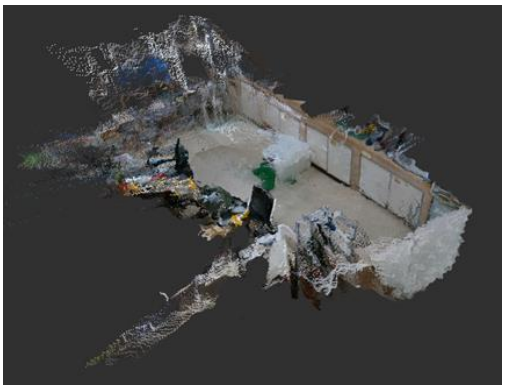

Fig. 6. Result of RTAB-Map built using laptop of the environment.

to integrate RTAB-Map onto our system if we have a better companion computer.

F. YOLOv4

YOLO or you only look once is a state-of-the-art, real-time object detection system. It is known for its speed and accuracy. Our original plan was to use YOLOv4 as our object detection method. However, due to the lack of enough dataset of lost properties, we still cannot obtain a good detecting accuracy after training before the demo day. Therefore, we consider that it should be possible to replace the find_object_2d package with YOLOv4 for better accuracy and more stability during object detection.

IV. Experiment Setup

There are multiple sensors mentioned above equipped on our robot. The two most important sensors of our project are the RealSense camera and the RpLidar. RealSense D435 camera is a RGBD camera used for 3D-object detection. The RpLidar is a 2D, 360 degree lidar with a range of 0.15 meters to 6 meters. It is used for both constructing the map of Gmapping and the navigation of move_base algorithm.

The following table is the spec of all sensors we used:

Table 1. Specification of the sensors equipped on the robot

Item	Specification
Jetson Nano B01	• GPU : 128-core NVIDIA Maxwell™ • CPU : Quad-core ARM® A57 CPU • Video Encoder : 4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265) • Video Decoder : 4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30| (H.264/H.265) • RAM : 4 GB 64-bit LPDDR4 • Storage : microSD • USB : 4x USB 3.0, USB 2.0 Micro-B • Connectivity : Gigabit Ethernet, M.2 Key E • Others : GPIO, I2C, I2S, SPI, UART
(Motor Driver) Stm32f103rct6	• Program Memory Size: 256 kB • Data Bus Width: 32 bit • ADC Resolution: 12 bit • Maximum Clock Frequency: 72 MHz • Number of I/Os: 51 I/O • Operating Supply Voltage: 2 V to 3.6 V
GY-85 IMU	• 9DOF : Sensor Module — 9 axis modules • Ultralow power : as low as 23 μA in measurement mode • Supply voltage range : 2.0 V to 3.6 V

RPLIDAR A1	• Measuring Range : 0.15m - 12m
	• Sampling Frequency : 8K
	• Rotational Speed : 5.5Hz
	• Angular Resolution : ≤1°
	• System Voltage : 5V
	• System Current : 100mA
	• Power Consumption : 0.5W
	• Output : UART Serial (3.3 v level)
	• Angular Range : 360°
	• Range Resolution
	  ○ ≤1% of the range（≤12m)
	  ○ ≤2% of the range（12m～16m)
	• Accuracy
	  ○ 1% of the range（≤3 m）
	  ○ 2% of the range（3-5 m）
	  ○ 2.5% of the range（5-25m）

Realsense D435	• Ideal range: 0.3 m to 3 m
	• Depth
	  ○ Depth technology: Stereoscopic
	  ○ Minimum depth distance at max resolution: ~28 cm
	  ○ Depth Accuracy: <2% at 2 m[1]
	  ○ Depth Field of View (FOV): 87° × 58°
	  ○ Depth output resolution: Up to 1280 × 720
	  ○ Depth frame rate: Up to 90 fps
	• RGB
	  ○ RGB frame resolution: 1920 × 1080
	  ○ RGB frame rate: 30 fps
	  ○ RGB sensor technology: Rolling Shutter
	  ○ RGB sensor FOV (H × V): 69° × 42°
	  ○ RGB sensor resolution: 2 MP
	• Connectors: USB-C 3.1 Gen 1

The following figure (Fig. 7) is the photo of our robot.

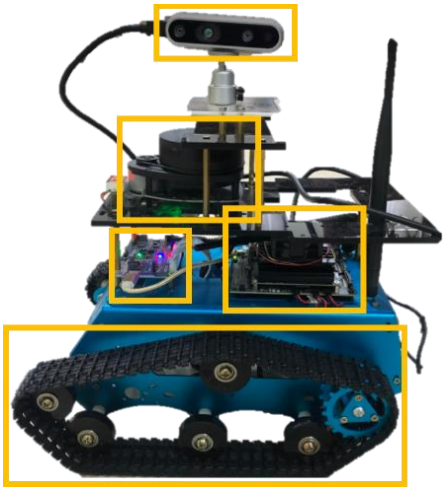

Fig. 7. Specification of robot, from top to down: Intel RealSense D435, RPLIDAR-A1, Jetson Nano, Driver/IMU board. We choose continuous tracks to cope with rugged terrain.

We can separate our experiment into three sections: environmental setup, object detection and guidance to object.

A. *Section I: Environmental Setup*

In this section, we first need to construct the map of the environment. The main algorithm we use for mapping is Gmapping, a SLAM algorithm highly relies on lidar. By using keyboard controlling we can move our robot around in a specific area, we can obtain the following map of the whole environment. This map will then be used for positioning the lost objects and to localize the robot's position. (See Fig. 8)

B. *Section II: Object Detection*

After building up the map of our environment, we can then request the user to upload the image of their lost property by using the find_object package GUI (See Fig. 9). The program will then find the feature points of the image using BRISK (Binary Robust Invariant Scalable Keypoints) algorithm.

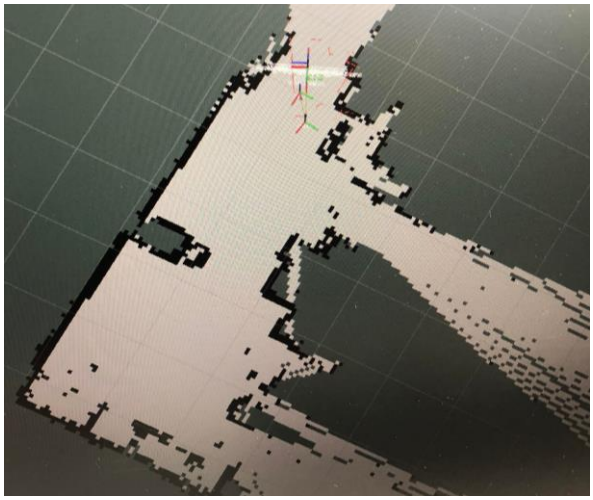

Fig. 8. Result of Gmapping map construction.

After multiple images of lost properties have been uploaded, the car is able to start to navigate automatically in the environment to find the objects uploaded by the user.

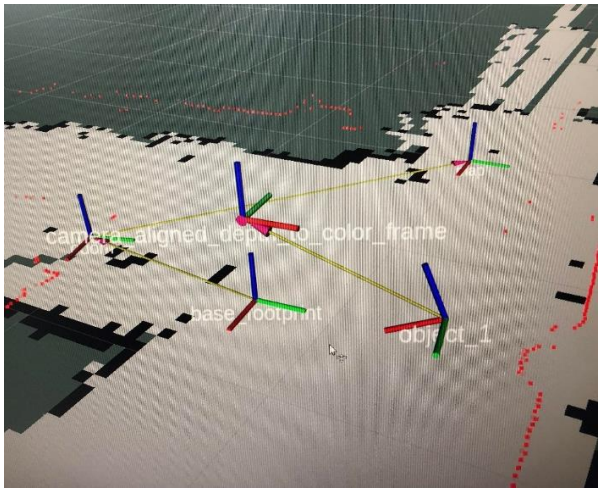

Fig. 10. The TF of the object is shown.

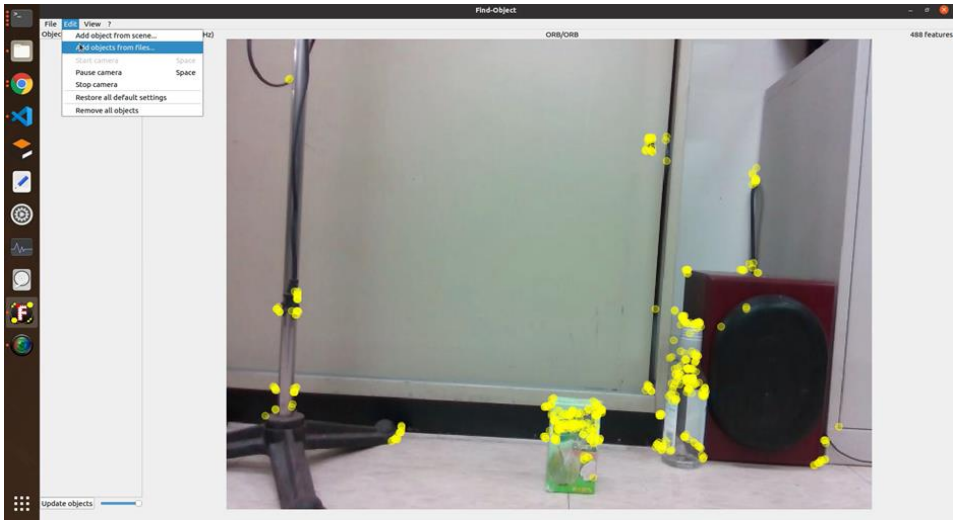

Fig. 9. The GUI of Find object 2d, the user can add object picture form cropping the current scene or upload a pre-taken picture.

Once a specific object is detected, the position of the object (with respect to the map frame) and the object ID will be stored into a list of our program. This list will then be used to guide the robot to the location of the lost property in the next section. Figure 10 shows that once the target is detected, the object frame can be viewed on the map (See Fig. 10). The object in scene would also be framed in find object GUI, as Figure 11 shows.

Fig. 11. The framed object in Find object 2d GUI

After navigating through the whole environment, the robot will go back to the origin of the map and wait for further commands.

C. *Section III: Guidance to Objects*

After the robot automatically navigates through the whole area, a list of lost properties is saved in our program. The user can then input the object ID (shown on the find_object GUI when uploaded image) through the command line (Figure 12.) to command the robot to lead them to the lost property. Notice that this was done by passing the object position to a ros topic move_base/goal. Therefore, the user can input multiple object IDs in sequence so that the robot will move to the destined point followed by the order of the user input.

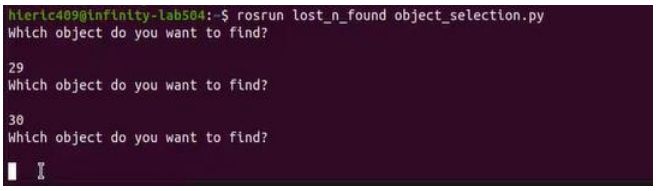

Fig. 12. Input interface for the user to select properties

V. Result

The goal of our robot can be divided into Three stages: manually map building using Gmapping, automatically object location searching/recording and user guiding by automatic navigation. After reached the goal, we would talk about the challenge we overcame.

A. *Stage1: Map building*

Figure 13 is the map we built. We marked several location to describe our environment setting.

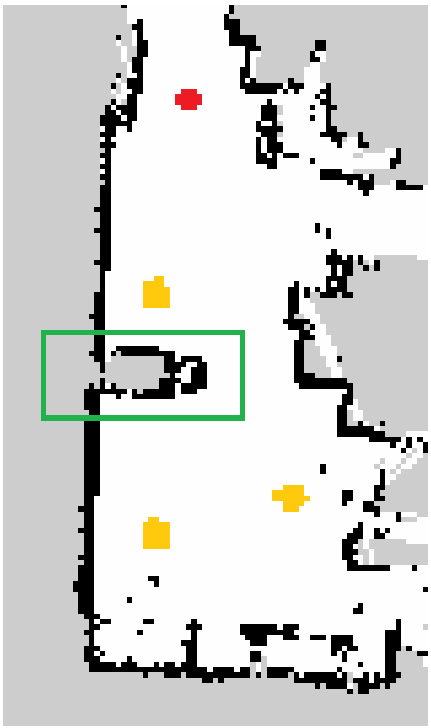

Fig. 13. Map and environment setting: red dot is the starting point of the robot; yellow dots represent the locaton of lost properties; green box indicates the location of obstacles.

B. *Stage2: Object Searching*

As we start the robot to search for lost properties, it moves to several corners and rotates to different orientations so that it can search in various views. After the robot detected the specific lost property, it will soon be framed on the find_object GUI and the object TF can also be seen on Rviz. In this stage, the object location will be recorded with the corresponding object ID (See Fig. 14).

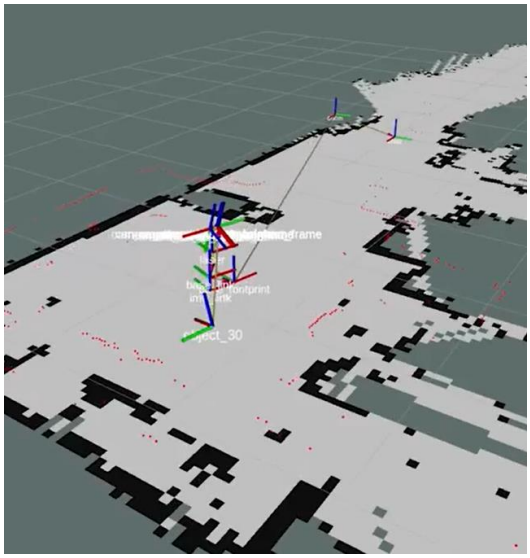

Fig. 14. The location of the object is recorded (Object_30).

C. *Stage3: Guidance to lost properties*

After all the lost properties were found, the robot heads back to origin. The terminal will then pop up a question to ask the user which object do they want to find. The user can type any of the object ID and the robot will lead them to the specific object by automatic navigation. The above command can be repeated to find all objects. In the end, if all the objected were found, the user can also order the robot to go back to the origin by typing "200" to the terminal.

D. *Challenge1: Datatype casting*

In python node, datatype is not defined until the node is execute. As a result, when the type is different (eg. UInt8 and String), the node would not publish anything, and also no error message to let us know the error. As soon as we find out this problem, we cast the data to meet the required datatype and solve the problem.

In C++ node, when datatype is different, it won't compile so we know the error beforehand. Thus, we cast them to meet the requirement.

E. *Challenge2: Topic pipelining*

In ROS, topic pipelining is important. Sometimes the subscriber and the publisher is not well connected, or the node does not even publishing topics. So we use rostopic echo command to check the pipelining is ok.

Another command we use is roswtf, it would look for broken pipeline and show how the pipeline should be connected. To check whether a node works correctly, we would use rosnode info to check the inbound and out bound topics the node subscribe to or publish.

F. *Challenge3: Image streaming latency*

At first, our remote find_object_2d node subscribe to /image_raw data. The latency cause the find object algorithm hard to work. As a result, at remote computer we transform the compressed image topic from the robot to raw image type, to reduce the latency.

G. *Challenge4: Power supply*

We tried to use the onboard battery to power the whole system. But the voltage drop when the battery is not fully charged would be nearly 1V and the Jetson Nano stops abruptly. As a result, we use cable to power up the car when testing, and we only use the battery in the demo.

VI. Conclusions

This project demonstrated the possibility of building up a robot car that can automatically find objects in a specific area and guide the user to the destined point. The robot was strongly viable to be further researched into a usable product which would be the next generation house furniture helping to cut down the time spent on Search and Sort. The first and simplest future work was probably installing a robot arm. The robot arm can not only bring the object to the owner after finding it but also be used for a more detailed scanning such as lifting other objects up to search for the target, just as humans would do to look from multiple viewpoints while searching.

Another thing to improve our project is to upgrade our companion computer. Jetson Nano which we currently use is definitely preventing us from completing the system as originally planned. Having a custom DSP and AI chip might be the best route to solve this problem actually. A custom chip would definitely decrease the power consumption, increase battery life, and also reduce price per performance.

For the software part, the future was literally the "original plan" in the introduction which included: 3D map object detection, which is an ongoing technology, implement RTAB-map on companion computers and implement CNN on object detection. And the recovery behavior is not ideal—the car would spin fiercely to match the orientation; we would consider to change the recovery behavior or loosen the threshold of "goal reached".

For the hardware part, our continuous tracks do not work quite smoothly—sometimes the wheels do not fit in the track so a bump is occurred. The bump caused the odometry of the robot to deviate. We would consider replace it with wheels if the terrain is flat enough to address this issue.

Acknowledgment

We thank Prof. Li-Chen Fu for letting us have the opportunity to conduct a term project like this. We thank CSIE Robotics TA group for helping us debugging and presenting. We thank Prof. Feng-Cheng Yang from IEE for providing the mobile platform.

External Links

Demo video link: https://youtu.be/7asykichNb4

Codes: https://github.com/liver121888/NTUCSIE-2021-Robotics-Assignments/tree/master/FP_team6

Work Division

Li-Wei Yang:

C++ node programing, TF tree design, integrate RealSense to chassis, launch file editing

Andrew Jian An Tay:

PPT presentation, RTAB-Map programing, code debugging, integrate RealSense to chassis, launch file editing

Hua-Ta Liang:

Video editing, RTAB-Map programing, code debugging, launch file editing, object_publisher node programming

Cheng-Yen Chung:

Operator for demo, Lost and found node python programming, object selection node python programing

REFERENCES

[1] Labbé, M, Michaud, F. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. J Field Robotics. 2019; 35: 416– 446. doi: 10.1002/rob.21831

[2] C. -Y. Wang, A. Bochkovskiy and H. -Y. M. Liao, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 13024-13033, doi: 10.1109/CVPR46437.2021.01283.

[3] A. Komarichev, Z. Zhong and J. Hua, "A-CNN: Annularly Convolutional Neural Networks on Point Clouds," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 7413-7422, doi: 10.1109/CVPR.2019.00760.

[4] S. Leutenegger, M. Chli and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," 2011 International Conference on Computer Vision, 2011, pp. 2548-2555, doi: 10.1109/ICCV.2011.6126542.

[5] S. Liu, "DragonFly+: An FPGA-based quad-camera visual SLAM system for autonomous vehicles", 2018 Artificial Intelligence Conference, 2018. [Online]. Available: https://conferences.oreilly.com/artificial-intelligence/ai-eu-2018/public/schedule/detail/69652.html

[6] Bay H., Tuytelaars T., Van Gool L. (2006) SURF: Speeded Up Robust Features. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11744023_32

[7] Pauline C. Ng, Steven Henikoff, SIFT: predicting amino acid changes that affect protein function, Nucleic Acids Research, Volume 31, Issue 13, 1 July 2003, Pages 3812–3814, https://doi.org/10.1093/nar/gkg509

[8] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," 2011 International Conference on Computer Vision, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544.

[9] Thrun, Sebastian, and Arno Bücken. "Integrating grid-based and topological maps for mobile robot navigation." Proceedings of the national conference on artificial intelligence. 1996.

[10] Fox, Dieter, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance." IEEE Robotics & Automation Magazine 4.1 (1997): 23-33.

[11] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, 2011, pp. 155-160, doi: 10.1109/SSRR.2011.6106777.

[12] Shim, V.A., Yuan, M., Tan, B.H., 2017. Automatic object searching by a mobile robot with single RGB-D camera, in: .. doi:10.1109/apsipa.2017.8282002

[13] Astua, C., Barber, R., Crespo, J., Jardon, A., 2014. Object Detection Techniques Applied on Mobile Robot Semantic Navigation. Sensors 14, 6734–6757.. doi:10.3390/s140406734

[14] Sjo, K., Lopez, D.G., Paul, C., Jensfel, P., Kragic, D., 2009. Object Search and Localization for an Indoor Mobile Robot. Journal of Computing and Information Technology 17, 67.. doi:10.2498/cit.1001182