國立清華大學 電機工程學系

實作專題研究成果報告

基於 RTAB-MAP 的室外自動導航系統的實現

Realization of Outdoor autonomous navigation based on RTAB-MAP

專題領域：系統組

組　　別：A90

指導教授：邱偉育

組員姓名：許晏誠、何立平、張瀚

研究期間：2020 年 7 月 1 日至 2021 年 5 月底止，計 11 個月

Abstract

Self-driving has been a prevailing topic of research for years. Recently with the introduction of Tesla, the realization of this technology has been shown promising. Autonomous driving requires knowledge of several fields, such as computer vision, localization, mapping and navigation. With the previous research and implementations, the indoor autonomous applications have shown some good results due to the simplicity of the environment. While indoor implementations have achieved a relatively complete framework, outdoor application has faced some hindrances, for instance, the stability of the sensors for outdoor uses and the complex environment which induce a large amount of environmental information and noises and the high cost of high precision sensors. Thus, in this paper, we'll present a simple framework of autonomous driving in an unknown place for further research on real-world on-road implementation.

SLAM and Navigation are two fundamental problems in self-driving, the former defines where the robot is in the real world while the latter tells the robot how it should do with a given goal. With the SLAM algorithm, the robot can have access to the environmental information of the surroundings, which then can allow the robot to have interaction with the real-world. The Navigation which consists of path planning and motion planning can make decisions for the robot to achieve a certain goal based on the information from SLAM. Among all the existing algorithms, RTABMAP has become our choice for mapping the real-world and localization in it. RTABMAP is capable of integrating multiple sensors' data and also supports, mapping mode and localization only mode together. TEB_local_planner works as our path planner for navigation with the ROS built-in navigation stack. Hardware setup is constructed by adding Realsense D435i depth camera and RPLIDAR A2 scanner as the sensors on a RC four-wheeled Ackermann car model. Camera is used as the main input for odometry and mapping while low cost lidar allow us to do ICP refinement and obstacle detection 360.

The proposed framework of system integration of various existing algorithms and hardware yields an outdoor self-driving platform for Ackermann vehicles which can perform mapping and localization with an acceptable precision and capability of navigation from point to point with obstacle avoidance. This platform can further be enhanced with other features like lane/sign detection and other behaviors for preference usage or sensor integration like high precision GNSS systems.

Our result showed that high precision localization and navigation with low cost sensors is not easy to achieve by single method. However, by integrating various algorithms and fine-tuning, it can be promising that the autonomous platform can be used to perform carrying purposes in a large-scale environment like campus.

摘要

自動駕駛的研究已經盛行好幾年，最近隨著 Tesla 的推出，該技術的實現已被證明很有前景。自動駕駛需要了解多個領域的知識，像是電腦視覺、定位、建圖以及導航。在之前得研究中，室內自駕的實施效果良好因為環境相對簡單。雖然室內已實現了相對完整的框架，但室外應用面臨一些障礙，例如室外傳感器的穩定性、環境中有大量信息和雜訊、高精度傳感器的高昂成本。因此在本文中，我們將介紹一個在未知地方進行自動駕駛的簡單框架，以進一步研究在如何在現實世界中上路。

SLAM 和導航是自動駕駛中的兩個基本課題，前者定義了機器人在現實世界中的位置，而後者則告訴機器人如何走到指定位置。使用 SLAM 算法，機器人可以取得周圍的環境信息與現實世界互動，導航由路徑規劃和動作規劃組成，可以根據 SLAM 的信息抵達目的地。在所有現有算法中，RTABMAP 已成為我們的選擇因其能夠集成多個傳感器的數據，並且同時支持建圖模式和僅定位模式。 我們使用 TEB_local_planner 及 ROS 內建的導航包來實現導航。至於硬體設置，我們在四輪 Ackermann 汽車模型上添加 Realsense D435i 深度相機和 RPLIDAR A2 光打。照相機用作里程計和製圖的主要輸入，而光達使我們能夠進行 ICP 精良和 360 度障礙物檢測。

我們所提出的將現有演算法與硬體做系統整合的框架提供了一個是用於 Ackerman 模型的室外自動駕駛平台，此平台可實現在室外具有一定精度的建圖與定位功能同時可以達成點對點的自動導航與障礙物閃避。該平台可以通過其他功能（如車道/路標檢測）和其他用於偏好使用的行為或傳感器整合（如高精度 GNSS 系統）來進一步增強。

我們的結果證明，採用低成本的傳感器很難實現高精度的定位和導航。
但是，通過整合各種算法和改良，可以有望將自走車平台用於在校園等大型環境。

Content

List of Figures/Tables

I. Introduction

Autonomous navigation has been a field of various technologies being researched. Including the study of SLAM, path planning and robot controller. Navigation within an unknown place is a crucial task in autonomous study. The robot learns information about the surroundings by processing the sensor's data via SLAM approaches which allow it to map the environment and localize itself within the map. In the past, most SLAM approaches were based on the data from laser range finder or Lidar for the High-Definition map of the real world. However, the high cost of 3D lidar is stalling further application of autonomous vehicles.

Recently, with the increasing computing power of the chip, processing complex images to obtain rich information has become practical. Thus, research and applications of implementing low-cost camera sensors on SLAM have emerged. For the past few years, implementation of visual SLAM has been conducted in this project. However, only the indoor environment has been tested and the navigation system was also not integrated. Most of the SLAM solutions on the market are for indoor environments, thus, our work of a self-built car-like robot suitable for outdoor usage should provide new probabilities to utilize the autonomous technique in a campus environment. Several barriers exist for the outdoor environment, like unclear edges of roads, shifting brightness with shadow and moving objects in the real-world situation. In this paper, we proposed an implementation method of current open-source Visual SLAM and integrate it with the navigation system for outdoor purposes. We aim to follow predecessors' work for indoor purpose, then finish with a functional system capable of both SLAM and navigation.

Our hardware model is based on a RC remote car equipped with a stereo camera with a built in IMU sensor and a low cost 2D lidar. Stereo camera is used to estimate the pose and capture the data for building the map. The lidar is used to provide a steady source of obstacle detection while autonomous navigating. The RC car is controlled by sending signals to the remote controller. The software model is a combination of RTAB-MAP and ROS Navigation stack with certain modifications of plugins. The final architecture plays as a pioneer that successfully combined several systems with navigation.

Our current result shows that the implementation is successful to a certain extent. The robot can navigate from point to point within a mapped space with the ability of obstacle avoidance. Still, however, parts of the system can be improved, such as the reaction time of avoidance, oscillations, etc. This paper can be organized into three main parts. In Section II, the underlying methods we used in the proposal will be briefly introduced including their theorem and architecture, the reason we chose them, etc. Section III talks about the architecture of our implementation and how we set the parameters or modify the system. Section VI shows how our experiment environments are chosen and what's the performance of our robot in different aspects under different conditions.

II. Background

A. Visual SLAM(V-SLAM)

V-SLAM aims to tackle the task of mapping and localization in an unknown region. The Visual SLAM method has been an emerging solution to the low-cost SLAM approach. Unlike 3D laser range finders or lidars which cost up to tens of thousands of dollars, sensors for visual SLAM cost only thousands of dollars. Typical sensors for

VSLAM are monocular, stereo and RGB-D cameras. In this section, basic architecture of a V-SLAM system and two state-of-the-art, RTABMAP and ORB_SLAM2, will be introduced.

1. Hierarchy of V-SLAM

Classic V-SLAM models can be roughly divided into five parts which can be seen in Fig. 1, Initialization, Odometry, loop closure, optimization and Mapping. A SLAM process starts with setting up a global coordinate system. Initialization of the global coordinate system enables the system to correctly identify the sensor location within the coordinate system. Next, the input data, which is the image from the camera, will be used to extract the feature points. With the feature points, images from different time stamps will go through a matching process to compute the camera pose within the system coordinate. The above operations produce the visual odometry that provides the information of the camera's motion.

Visual odometry(VO) is commonly integrated with an inertial measurement unit(IMU) for higher precision. In the meantime, loop closure will be performed to check if the place is visited. If a place is found to be visited, the information will be sent to back-end optimization together with the real-time VO data. The optimized pose estimation will then be utilized to locate the camera within the system coordinate while the mapping will be processed through methods like projecting the point cloud created from the images' feature points. Two state-of-the-art, ORB_SLAM2 and RTABMAP will be considered in the following sections.

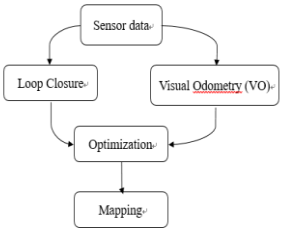


Fig. 1. Basic blocks of V-SLAM method

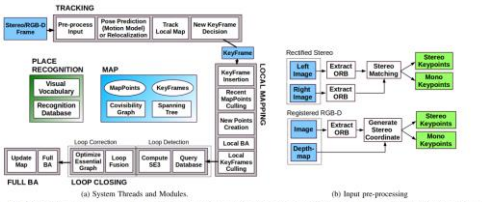


Fig. 2. Procedure of ORB-SLAM2

2. ORB_SLAM2

ORB-SLAM2 is a complete SLAM solution for monocular, stereo and RGB-D cameras which is capable of map reusing, loop closure detection and relocalization. ORB-SLAM2 provides a stable and high accuracy localization by utilizing several state-of-the-art algorithms. Besides SLAM mode, ORB-SLAM2 also supports localization only mode if the environment has already been mapped. The method rolls out two important features that makes it stand out from other SLAM solutions. First, Bundle Adjustment(BA) induced higher accuracy than those high performance methods based on ICP or photometric and depth error minimization. ORB-SLAM is composed of three blocks, which are tracking, local mapping and loop closing respectively as shown in Fig. 2.

The three procedures run simultaneously. The main task of tracking is to pre-process the image by extracting ORB feature points and estimate the pose according to the last frame. If the pose cannot be estimated, then the system will perform relocalization which will reinitialize the pose and track the local map to decide new key frames. LocalMapping constructs the local map by inserting the keyframes and eliminating some recent map points while creating new ones. After creation of new map points, local BA will be performed followed by culling some of the key frames.

Lastly, loop closing will be carried out. Loop closing has two parts, loop detection and loop correction. First, a loop has to be detected and secondly the loop will be corrected optimizing a pose-graph. ORB-SLAM2 uses stereo/depth information which results in less drifting. After loop closing, full BA will be conducted and the map will be updated.

3. RTABMAP

RTAB-MAP, for Real-Time Appearance-Based Mapping, is a visual SLAM method whose core concept is based on loop closure detection with memory management to overcome the large-scale and long-term online operation. RTABMAP tends to meet the requirements as the following: online processing, robust and low drift odometry, robust localization, practical map generation and exploitation and multi-session mapping, kidnapped robot problem. Next, the structure of RTAB-MAP is going to be discussed.

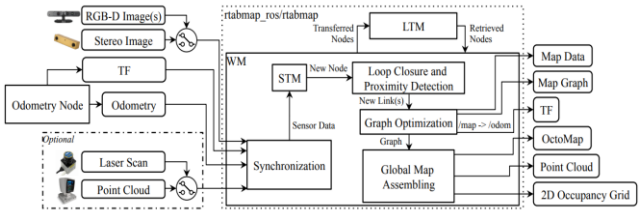


Fig. 3. block diagram of RTAB-MAP in ROS nodes

3.1 Structure of RTAB-MAP

As we can see in Fig. 3, which shows the block diagram of RTAB-map in ROS nodes, RTAB-MAP consists of several independent part: the sensors, TF, odometry and rtabmap itself which perform the loop closing, graph optimization and map building. This feature of separating the blocks allows it to be more diversely compatible with other open-source algorithms, and also makes it easier to compare different setups of sensors or algorithms. Basic concept of RTAB-MAP is that the data that optimization and mapping needs, like sensor data, both images or pointcloud, and odometry and TF information, will be synchronized in the working memory.

All operations will be done in working memory(WM) with a limiting size, thus, a long-term memory(LTM) will be used to store the nodes that aren't currently being used. With this approach, the memory of the database in WM won't unlimitedly increase, however, the decision of which nodes are going to be saved to LTM becomes crucially important. In the following sections, two nodes that are the most important while utilizing RTAB-MAP will be explained.

3.2 TF Node

TF stands for the transformation between the coordination of the "sensor_frame" and the "base_link" where "base_link" refers to the coordinate system of the main robot. TF is a ROS package that can make users easily create a correctly fixed coordinate system for rtabmap to do the localization and mapping. A wrong TF can lead to misleading orientation and odometry thus causing the navigation to malfunction or the map being unrecognizable.

3.3 Odometry Node

Odometry can be seen as the basis of the SLAM method. It provides the estimated pose of the robot in the form of an Odometry message with the corresponding TF's transform (e.g., /odom → /base_link). This node can implement any kind of odometry

approaches while some of them have already been integrated in the RTAB-MAP, such as VISO2, ORB2, VINS-Fusion. After building RTABMAP with them, they can be used directly as internal odometry. On the other hand, external odomery like wheel encoder, imu or other approaches can be implemented by subscribing to the topic from external odometry sources. Meanwhile fusion of different odometry can also be done loosely with the "robot_localization" package which utilizes EKF filters for localization. However, since each source is only loosely bonded, the performance will only be high if all the sources are calibrated. While the loop closing is based on mostly visual data, odometry can be provided via both visual and Lidar sources which will be discussed later.

3.3.1 Visual Odometry

Default stereo and rgb-d Visual odometry(VO) behaviors are shown in Fig. 4. First, feature extraction and matching will be done with the image and TF input. Then, a series of motion estimation and prediction will be processed with local BA used. After computing, the posee will be updated and sent out as odometry and TF sources while key frames will be added conditionally.

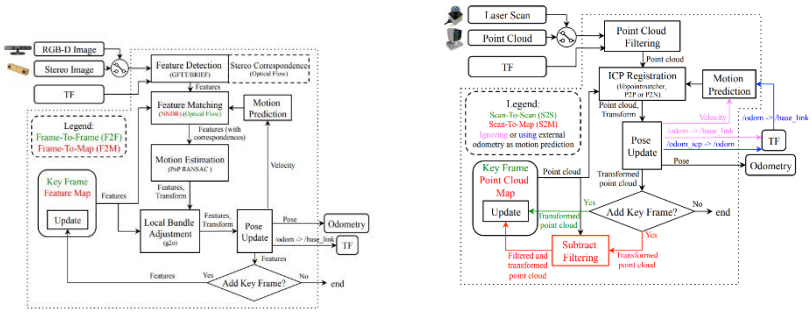


Fig. 4. stereo & rgb-d odometry using F2F and F2M approaches & lidar odometry

3.3.2 Lidar Odometry

Lidar provides a point cloud that can be used directly through filtering, thus, the computation resources for lidar odometry is lesser. After the pointcloud is filtered, pose is estimated by ICP registration. Then the pose will be updated and sent out while the key frame will be added like visual odometry. However, the information for a lidar is not as rich as a camera, thus, lidar odometry is mostly tested within a stable indoor environment. The block diagram is shown as Fig. 4.

3.3.3 Visual Inertial Odometry (VIO)

Since visual odometry fully depends on the input visual images for computation, and can only be refined when a loop closure is detected, in a dynamic or plain environment without sufficient visual features, VO approach will fail. On the other hand, imu odometry will have incremental error after a period of time and cannot be refined automatically. Visual and IMU approaches actually compensate for each other, that's why the VIO approach is rolled out. Approaches like OKVIS and VINS-Fusion can be chosen as an odometry source for higher accuracy.

B. ROS Navigation

In the field of the autonomous vehicle, navigation plays a grave part. Navigation not only gives the rough route toward the destination, but gives detailed directions of how to get to the destination and the control of the robot during the process. Hence, a fine-tuned and precise navigation method is required for a self-driving car to operate.

In our proposed implementation, we utilized the navigation stack with the ROS system. The navigation stack was initially built for differential robots, for instance, TurtleBot.

However, with some extra plugins which meet the format of the navigation stack we've found, navigation for ackerman robot, like a car-like robot, is practical to do. Navigation stack provides a framework with a standard dataflow that allows us to integrate the plugins and external software easily. The main structure of the navigation stack is the "move_base" package in Fig. 5, which contains the planner and recovery behavior and other foundations to give commands to the robot. The details will be discussed below.

1. Framework

From Fig. 5, we can have an overview on how the navigation stack works. First, the static from the previous SLAM approach needs to be fed in as the global costmap. Second, the sensor data is required to show the real-world environment like obstacles on the costmap. Third, a goal is given through RVIZ, then the global planner will produce a global path based on the global costmap. Lastly, the local planner will compute the exact trajectory for the robot relying on the real-time local costmap and the global path. Different blocks in the navigation stack will be introduced.

2. Inputs/Outputs of Navigation Stack

In Fig. 5, we can see the default input topics for navigation stack are '/goal', '/tf', '/odom', '/map' and sensors' topics. Goal is the coordinate of the destination on the global costmap, which is the map from the SLAM approach. TF topic is built up with several sources, amcl, sensor transform and odometry source. AMCL is the localization package by default. It provides the coordinates of the robot in the static map from map_server. Sensor transform is the relative position between sensors and the robot's base_link like Fig. 6 shows. It can be defined through URDF or using the static transform package directly. Odometry source is mentioned in "RTABMAP - Odometry Node". It provides the trajectory of the robot in the real world. Odom topic is directly fed from odometry sources. Map topic contains a static map that can be a pre-built map sent from map_server package or a real-time map from SLAM. such as /grid_map of RTAB-MAP. The static map will be considered as the global costmap in the move_base package.

Last, sensors' topics have the sensor data in two formats, LaserScan and PointCloud. The data will be used to present the obstacles in the environment on the Costmap. The output message is sent from the topic "cmd_vel". It's a Twist message format that is represented in the form of linear velocity, x and y, and angular velocity, z. The subscribing node for "cmd_vel" is the base_controller. The controller will process the twist message to signal that the hardware can understand with additional constraints like PID constraint.

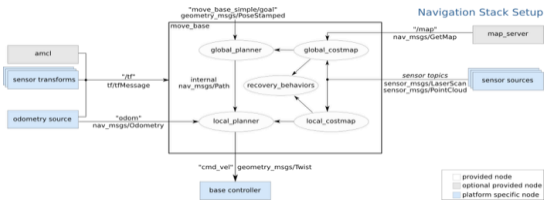


Fig. 5. Illustration on how to use navigation stack

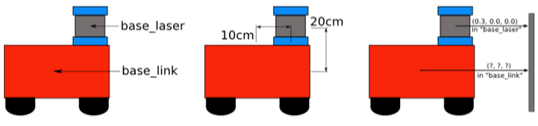


Fig. 6. Tf example of a robot

3. Move_base

Move_base is the core of the navigation stack, which includes three important features, planner, costmap and recovery behaviors. Move_base can process all the input data and come up with a feasible trajectory for the robot.

3.1 Costmap

Costmap is the fundamental element for path planning. We can consider it as the map that the robot will see. It holds the information of the obstacles in the form of grids on the map. Global and local costmap are used by global planner and local planner respectively. The costmap needs to be clearly set via the yaml files in order to provide the correct information of the surroundings.

3.2 Global Planner

Global planner will read the coordinate from the '/goal' topic and compute the global path based on global costmap. Several methods have been proposed to find the optimal global path for time efficiency or distance efficiency. Such as the simplest planner, carrot planner. It checks if the given goal is an obstacle, and if so it picks an alternative goal close to the original one, by moving back along the vector between the robot and the goal point. Here, we used the Global_Planner which is the refinement of Navfn. Previously, Navfn used Dijkstra's algorithm to find a global path with minimum cost between start point and end point. The later Global_planner is built as a more flexible replacement of Navfn with more options. These options include 1) support for A* algorithm 2) toggling quadratic approximation 3) toggling grid path.

3.3 Local Planner

The local planner receives the global path from the global planner. It tends to follow the global path but compute the detailed movements depending on the capacity of different robots. For example, for differential robots the local planner will have a local path that tells the robot to turn in place while for a car-like robot it will require a trajectory that performs a turn with an arc of turning radius. The default local planners are base_local_planer and DWA_local_planner. The base_local_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. DWA is the rewrite of the dwa option for base_local_planner.

The algorithm is shown in Fig. 7. The DWA performs a sample based optimization. It samples a control action in the feasible velocity space (usually a translational/angular velocity pair) and rolls out the trajectory for these particular sampled actions. Since both of them only support differential robots, we've resorted to TEB_local_planner which supports car-like robot movements though it's still experimental.

The underlying method for TEB called Timed Elastic Band locally optimizes the robot's trajectory with respect to trajectory execution time, separation from obstacles and compliance with kinodynamic constraints at runtime. In order to avoid situations of stucking in a locally optimized trajectory. The method also utilized the concept of homotopy classes which enable the robot to store some candidate paths as backup. The comparison of DWA and TEB is shown in Table. I.

DWA	TEB
· Suboptimal solutions without motion reversals (control actions are kept constant along the prediction horizon) · Well-suited for diff-drive/omnidirectional robots, but not for car-like robots · Supports nonsmooth cost functions	· The planned trajectories are closer to the actual optimal solution, but constraints are implemented as penalties only. · Suited for all robot types · Planning of multiple trajectory candidates in multiple topologies · Dynamic obstacle support (experimental) · Large computational burden

Table. I. Comparison of DWA and TEB local planner

3.4 Recovery Behavior

When the trajectory from the local planner is not feasible in real-time, the move base will turn to recovery behavior to escape from the current state to compute another trajectory. Default behaviors of move_base are ClearCostmapRecovery and RotateRecovery. The former attempts to clear the space by reverting the costmaps used by the navigation stack while the latter tries to rotate the robot 360 degrees if local obstacles allow it to find where the path and obstacles are again. Rotate recovery is suitable for differential robots but not car-like robots. Thus, we replaced the rotate recovery with a recovery approach called 'stepback and steerturn recovery' provided in [7]. The flow of recovery is illustrated in Fig. 8 in four steps. First, a robot stucks, which disables the robot from turning in any direction. Second, it attempts stepping back for space to turn with steering. Third, the robot scans the navigation stack's costmaps at the point where the robot stepped back. Last, it detects the direction to the nearest obstacle and attempts to turn in the opposite direction.

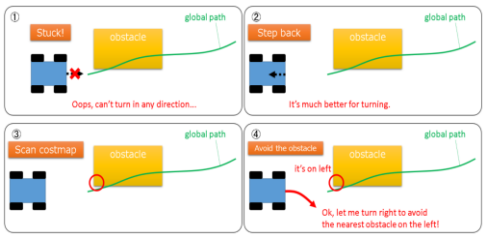
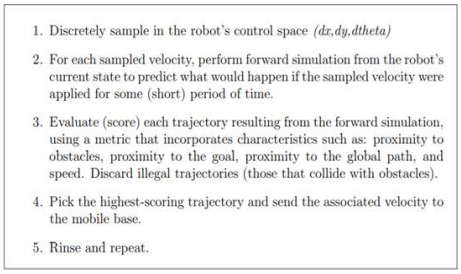


Fig. 8. Illustration the flow of stepback_and_steerturn_recovery



1. Discretely sample in the robot's control space *(dx,dy,dtheta)*
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.

Fig. 7. DWA algorithm

III. System Implementation

A. Hardware Setup

1. Robot configuration

1.1 ackerman steering geometry

The underlying steering model for our robot is the "ackerman steering geometry". This model is commonly used in ordinary cars, so it's also called a car-like model. We chose this steering system because car-like models are the most common one on the road. Once the system for ackerman is integrated, there'd be no need to put extra effort on developing differential models. Second, a stable but large enough model to meet our purpose is hard to find, so we'd prefer the Ackerman model as our choice. The following Fig. 9 shows how Ackerman steering geometry works.

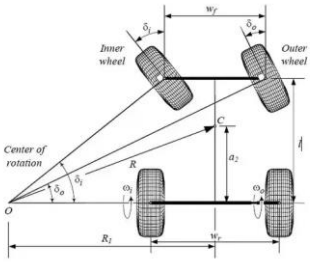


Fig. 9. Ackerman steering geometry

Fig. 10. Robot model

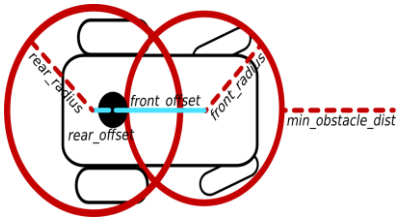


Fig. 12. Footprint model of type "two circle"

1.2 Robot architecture

The model of our hardware system is based on a four-wheel drive RC remote car. Four-wheel driving and a large storage area enabled us to lay our laptop which works as the computing core, on the car. Sensors are set up according to the tf including a RGB-D camera which can also perform as a stereo camera and a 2D lidar which allows the robot to detect the obstacle 360. Fig. 10 shows our robot model containing a Lidar on top and stereo camera at the front.

1.3 Sensor configuration

1.3.1 Depth Camera

"Depth cameras" are cameras that are able to capture the world as a human perceives it – in both color and "range." The ability to measure the distance (aka "range" or "depth") to any given point in a scene is the essential feature that distinguishes a depth camera from a conventional 2D camera. Depth cameras can use any visual features to measure depth, they will work well in most lighting conditions including outdoors. The Depth camera we are using is Intel® RealSense™ depth camera D435i. D435i also contains an inertial measurement unit (IMU), so it can calculate the movement of our robot.

1.3.2 2D Lidar

Lidar is a method for determining distance by targeting an object with a laser and measuring the time for the reflected light to return to the receiver. Lidar is commonly used for autonomous cars to detect the distance of obstacles and roads. The lidar we are using is RPLIDAR A2. RPLIDAR A2 is a low cost 360 degree 2D laser scanner(LIDAR). The system can perform 2D 360-degree scan within a 16-meter range. The generated 2D point cloud data can be used in mapping, localization and object/environment modeling.

2. Controller design

First, there is a signal called "cmd_vel" which is published from the ROS navigation stack. "cmd_vel" contains the linear speed and the angular speed of the robot. Then, the twist message will be encoded to an integer waiting to be sent. After the Arduino board receives the encoded "cmd_vel" signal, it'll decode the signal back to the original value. After the calculation, the Arduino board will output four kinds of signal(forward, backward, left, right) to the remote controller. Since the control is currently restricted to the remote controller, the updating frequency of our cmd_vel cannot be too fast. Otherwise, the remote controller will not be able to handle the signal. Thus, in navigation setup, we constrain the controller frequency to 2Hz. Fig. 13 shows the command flow of the Arduino.

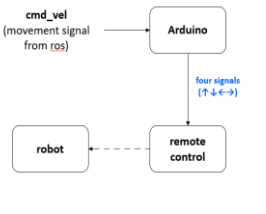


Fig. 13. Command signal flow

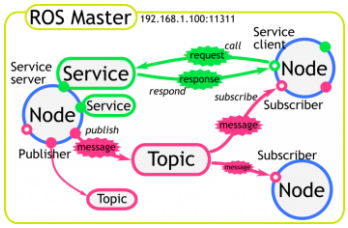


Fig. 14. relationship of Node, Topic and Service in ROS

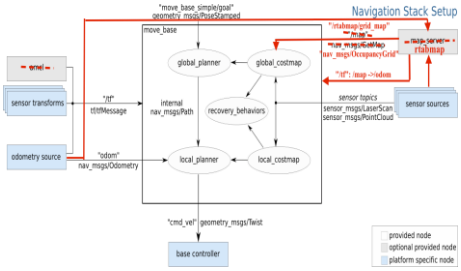


Fig. 11. An overview of RTAB-MAP using navigation stack

B. Software Setup

Our system is based on Ubuntu 18.04 and Robot Operating System(ROS). ROS has now been the largest open-source platform for researchers to implement robot related algorithms with a standard data type and stream flow. ROS has a core master where the system will operate around it. Topics of data will be published and subscribed by nodes and the data can easily be monitored during the experiment. ROS has become our fundamental basis of integrating all the system software we'd like to test. Although ROS is the top choice platform for us to do academic research on, it is actually not suitable for real-time commercial usage. Since all systems work around the ROS core master, once the master is down due to some malfunction, the whole system will be malfunctioning. We've experienced some situations where certain nodes are not working but rebooting the whole system will solve the issue. Fig. 14 shows how the ROS works by connecting nodes.

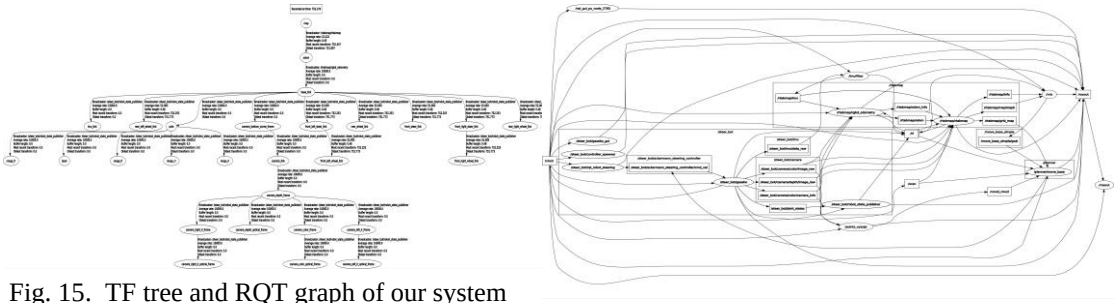


Fig. 15. TF tree and RQT graph of our system

1. RTABMAP

ORB-SLAM2 was initially our first choice of SLAM approach due to its accomplishment of being state-of-the-art in the field of visual SLAM. However, for the reason that we cannot produce an occupancy grid map for navigation through the gui of ORB-SLAM2, and the RTAB-MAP allows multiple sensor types' input along with the capability of allowing external odometry sources, we chose RTAB-MAP as the final decision for mapping and localization task. RTAB-MAP and ORB-SLAM2 both perform great with large-scale outdoor environments. How RTAB-MAP is set up in our system and what role does it play will be briefly introduced in the following context.

1.1 RTAB-MAP installation

In order to use ROS in our future application, two versions of RTAB-MAP will need to be installed, which are standalone versions and rtabmap_ros respectively. The former is the main structure of RTAB while the latter is the API for RTAB-MAP to be used in ROS. While building a standalone version of RTAB-MAP, it is necessary to build it with GTSAM and VINS-Fusion where the former is the external loop closure algorithm and the later being the external VIO source. To ensure that both of them are built

together with the RTAB-MAP standalone version, check if the cmake configuration shows both of them ON. Rebuilding the workspace where rtabmap_ros is is also required for the options of GTSAM and VINS-Fusion to be utilized with RTAB-MAP.

1.2 Localization

Localization plays the most important part in self-driving vehicles. The strategy of odometry and loop closing affects the quality of localization the most. We chose VINS-Fusion which takes stereo and imu data as our odometry source after some experiments with stereo, rgb-d and ICP odometry. After trials, we found that since the outdoor environment has too few obstacles for the Lidar to reflect and the sunlight has too large an impact on the RGB-D image, thus, these two odometry sources are not taken into consideration. Hence, stereo odometry (F2M) had been our top choice. Outdoor environments are rich in visual features so that stereo odometry can perform well, however, in some cases due to steep changes in brightness due to shadows can sometimes cause the odometry to drift suddenly. To prevent the inconsistent of consequent odometry due to unexpected changes in images, also it's difficult to have images of all orientation in an outdoor environment, IMU is considered to refine the drifting of odometry and jumping in localization mode which will cause redundant computation while navigation. The optimization of localization is enhanced by using the neighbor links from the laser.

1.3 Mapping

In RTAB-MAP, there are two options for the source of the map. By setting "/GridFromDepth" to True, we can obtain a 2D occupancy grid map by the depth image, otherwise we'll get an output from our Lidar laser data. For an indoor environment. The bounding of the map should be a solid surface which is easy for the laser to recognize. However, an outdoor environment doesn't actually have a clear boundary but some bushes or paveways as the edges. Thus, lasers cannot scan those static obstacles but cameras can see them, so our 2D occupancy grid map is produced from the depth data. RTAB-MAP produces the map by projecting the point cloud that is above the ground plane to the surface. Hence, having the camera facing the correct roll,pitch,yaw angle is important as well as defining the correct ground plane by adjusting the parameters under "/Grid". Defining reasonable obstacles is also grave for the system to have a right cognition of where it can go through, like a small tunnel. 2D mapping is performed in our implementation to save computation resources for navigation. Every mapping session creates a database which the data is stored in. If we have multiple databases that have some parts overlapped, we can merge these databases to obtain a full scale map of a large-scale environment.

1.4 Other Features

RTAB-MAP also supports another nodelet named "obstacle detection" which reads in the pointcloud then outputs the point cloud of the obstacles. We feed the pointcloud of the intel realsense d435i camera into the voxel filter to filter out light weighted points in order to reduce the computation time for the nodelet. Then, the filtered cloud will be fed to the nodelet to compute the obstacles cloud seen from the camera to topic "/obstacles_cloud" which will be utilized by the local costmap to compute the local path while self navigating. Second, RTAB-MAP provides a useful tool, rtabmap database viewer, this viewer allows us to check the recorded data during mapping along with the post-processing function. When a drift occured in the mapping session, we can

refine map path by post-processing in the database viewer. We can also edit the grid map manually if we want to set boundaries that aren't mapped.

2. Navigation

The navigation stack in our implementation is modified as in Fig. 11.The map_server node is substituted with rtabmap node which provides the static map via the topic "/rtabmap/grid_map". The odometry sources and sensor sources are also sent to the rtabmap node like previous sections mentioned. The TF here is published by the rtabmap node. What's worth mentioning here is that since the /map topic is published directly from rtabmap, the navigation can be done with mapping simultaneously, which we regard as exploration mode. However, in exploration mode, the efficiency will not be high for the reason that the map is not always static due to the drifting of odometry.

2.1 Global/Local Planner & Recovery Behavior

The default global planner for move_base is navfn, which has less parameters to tune. However, navfn only provides the coordinate information of the global path without orientation information. Without the orientation information, TEB local planner might result in an odd trajectory like moving toward the goal backward while the back of the robot doesn't contain a camera sensor but only lidar which cannot detect low-height obstacles. TEB is chosen as the local planner for its suitability for a car-like robot.

The priority of tuning TEB is to measure the footprint of the robot precisely, otherwise TEB will result in a weird and feasible trajectory including constant back and forth motion. We first set the footprint as a line. After observing a unintuitive route with the robot stuck in a series of oscillations, we changed the footprint type to "two circles". The radius of the footprint circle with addition of min_obstacle_dist is the minimum distance between the steer and rear axis and the obstacle. The footprint model is shown in Fig. 12. The weight for forward drive is also set high to prevent constant backward movements. Homotopy class is turned true to prevent stucking in local optimal trajectory while the computation is increased. The final trajectory is presented in the format of the Twist message which will be received by the arduino controller. The recovery in move_base is changed to stepback and steerturen recovery in the move_base.yaml file.

2.2 Cost map

The global costmap utilized the static map from rtabmap directly without modifying. The global planner will create the path based on the costmap. Since global costmap won't be modified during the navigation process, the mapping session of the environment is crucial. The mapping should be performed with the environment cleared out without any dynamic obstacle for the best quality map. If the environment has changed, the mapping session needs to be performed again, otherwise the global planner will not compute the optimal path or even a wrong path for the robot to crash.

As a result of that we use the global costmap for global path planning only and we don't want the global costmap to be changed constantly, no sensor data will be fed to the global costmap. Unlike the global costmap, local costmap is used to show dynamic obstacles in the real-time environment. Thus, lidar's laser scan data and cameras' point cloud data is sent to the local costmap. However, the point cloud directly from the camera is noisy which might lead to mis-recognizing the obstacles.

Our result actually showed that using the point cloud directly will cause the robot to consider the whole front sight as obstacles. Thus, a voxel filter needs to be passed first. Then, the filtered cloud will be sent to rtabmap obstacle detection node for

detection and showing the obstacles' cloud only. Last, only the obstacle will be subscribed by local costmap. The local costmap size is set to 3x3 with the robot at the origin. Costmap converter is also used to reduce the computation power.

IV. Experiment & Result

During the one-year project, two phases of experiment have been done. The first phase is where we utilized each part of the system to find the one that meets our requirement. This part was done only with a handheld stereo camera at first. Later on, the integration of the system was implemented in the Gazebo simulation, including the SLAM approach and Navigation system. In the second phase, we started to put the integrated system in practice on our RC model and test the performance in a real-world environment which sits within NTHU campus with further parameters and system tunings. Next, the two-phase experiment and result will be discussed.

A. System integration in Gazebo

After trying several benchmark large-scale worlds for autonomous driving, we couldn't find the one that our computing system can afford. Thus, we built a small scale environment from open source models which includes a maze, roads with lanes and obstacles on the road with houses on the side. The world is shown in Fig. 17. The maze allows us to examine the performance of mapping and localization in a hallway situation while the open space city scene enabled us to determine the performance within an outdoor environment. System based on RTABMAP is tested with several odometry sources and path planning algorithms.

1. Maze

In the maze environment, the visual feature is monotonic in which the robot will lose itself after a corner since it considers that it's still in the same hallway. Here, we can notice the defect of the visual SLAM. If the environment has similar visual sight. Thus, we've come to the conclusion that in an indoor space, or specifically a space with clear boundaries and less visual features, Lidar-based SLAM will be a more proper approach.

2. Open area

In the open space environment, the robot has relatively good odometry, however, since the simulated environment has less feature point, the map produced is not clear enough. The odometry was lost in a certain angle of sight for the emptiness of visual sight.

Overall, the navigation works well with high quality of odometry sources. If the odometry sources are noisy then the route of the robot will become noisy and oscillate a lot. To sum up, the integrated system of RTAB-MAP and Navigation works fine but with some flaws within the simulated Gazebo world. The simulated environment and results are shown in Fig. 17.

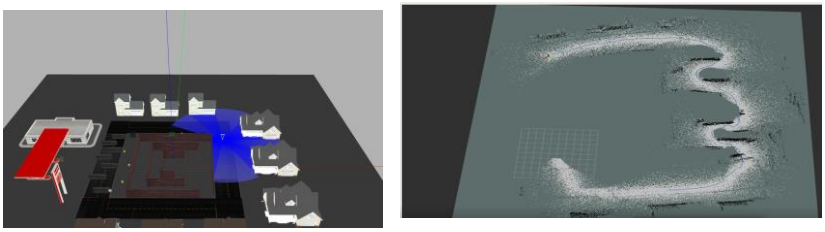


Fig. 17. Gazebo simulation world & corresponding map

B. Implementation in the campus

The test was held in two kinds of environment, open space and semi-open space.

1. Open space

Open space test took place around the delta building which is a square. By comparing the starting point and ending point also with the ground truth path, we found out that with stereo odometry in an outdoor environment, the error can be controlled within 30cm. However, several points need to be carefully considered.

First, the sun must not appear in the sight of the camera, otherwise the auto-exposure will make the area besides the sun black. To avoid this scenario, we set the ROI of auto-exposure within the lower side of view. Second, the same path might need to be recorded repeatedly in order to either fix the drift in odometry caused by the sudden change of brightness or facing a wide empty area by more loop closures or clearing the dynamic obstacles in mapping sessions like cars or pedestrians by revisiting the same place again. Third, the speed while mapping the environment can't be too fast in case that the computing doesn't catch up the speed.

Troubles have been encountered during both mapping session and localization with navigation session. With exploration mode on, e.g, The robot will do the mapping session and navigation at the same time, the path planning is varying quickly and out of the way in the real world. That's because to constrain the computation sources of mapping nodes, the range of maps being built is restricted while navigation stack requires the map to be provided. Second is that the time for localization mode initialization lasts for several seconds each time we reboot the system. Third, the height of the sidewalk is sometimes too low for the robot to recognize as obstacles.

Also, when performing backward behaviors, lidar cannot scan the height of the sidewalk causing the robot to be stuck at the place while the recovery behavior requires the robot to also perform backward motion. In future, forcing forward motion only needs to be achieved to act safely and intuitively on road. Fourth, since the controller frequency is retained to 2Hz, which is quite slow, the robot will not avoid fast moving obstacles like bicyclers in time. Fifth, our robot now only performs mapping and path planning within the whole empty space in the grid map. It won't take the lanes or road sign instruction into account. Integration of computer vision with controllers must be done to accomplish such an objective. Sixth, the mapping session consumes large memory, computation resources and power. Memory issues can be worked around by setting memory threshold, however, mapping the whole campus by human is still time consuming.

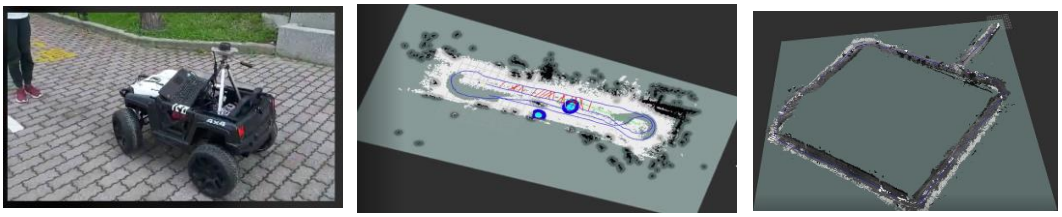


Fig. 18. Demonstration of obstacle avoidance with navigation in outdoor with the reference map

Thus, how to create a static map directly from a human-based map like Google Map while the robot can still localize within it is an important topic, meanwhile the RTAB-MAP requires the recorded database to perform localization. As a result, our proposed model has reached a goal that it can map and localize itself in an outdoor environment with enough light and also can perform self-navigation though with some defects like oscillation and the detection of obstacles like we mentioned above. The results of mapping and demonstration of navigating is shown in Fig. 18.

2. Semi-open space

Experiment focusing on navigation was held in a semi-open space within a square surrounded by hallways like in Fig. 16. The condition is set up with arbitrary static obstacles while mapping sessions. We can notice that there are unknown spaces around the obstacle due to the varying color of the ground. The ground with darker color will be considered as unknown spaces while white ground as ground in grid map. However, the problem can be circumvented by allowing the planner to make a plan within unknown spaces. During the experiment we've discovered several problems.

First, though the visual sight of the robot in different angles seems distinct for us, the robot actually grabs similar feature points in varying angles. Thus, we can observe some small jumps of odometry. These jumps can cause the global planner to recompute the path every time the odometry jumps which causes the car to perform a non-instinct behavior and also takes longer to get to the goal. Another issue is that when we set the goal at the back of the robot it will somehow be stuck at a local optimal path during the process. Stuck in a local optimal path means that it will plan a path that's going forward. After going forward the local trajectory will then become backward which causes the robot to be trapped in a constant back and forth motion.

In order to solve the trouble, we try to give a specific orientation goal by changing the global planner. With Navfn we don't have information for orientation, so the local planner will keep changing methods for the optimal path. Hence, with the refined Global planner, we can give specific orientation information to the robot which keeps the planner from changing orientation at the destination for the optimal path.

After trials of obstacle avoidance, our robot can now detect the obstacle through the filtered point cloud of the camera along with the one from LIDAR. When facing a moderate speed dynamic obstacle like a pedestrian, our robot can safely go around it without collision even if the obstacle keeps blocking the robot on purpose. However, while facing obstacles that's not on the same surface with the LIDAR nor in the sight of the camera, our robot can't detect them because the camera's sight is bounded and the LIDAR is in 2D only. The result can be seen in Fig. 16.

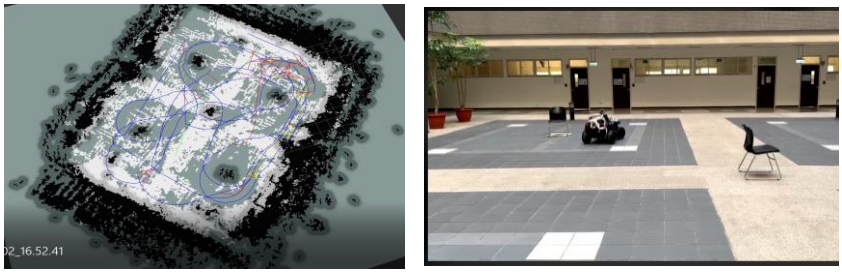


Fig. 16. Demonstration of obstacle avoidance with navigation in semi-open space

As a result, our model yields a successful outline for autonomous driving. It can fully perform mapping and localization in an unknown place. It is also capable of path planning and motion planning with a given destination on the map along with obstacle avoidance. The above result is tested in a complex outdoor environment with people passing by. However, the circumstance is still under a certain control like there's no car or fast moving obstacles for safety issues. Second, the tests were carried out under condition with sufficient light sources. Still, some aspects need to be further enhanced, like the ability of lane following or reducing the oscillation around the global path. To sum up, our proposed model yields a platform of outdoor SLAM based implementation of car-like UGV which can be the basis of future research and enhancements.

V. Conclusion

In this paper, we rolled out an implementation of system integration for an autonomous vehicle. The system provides an architecture which is composed of visual SLAM, RTAB-MAP, and ROS navigation stack with modification plugins. The proposed model is based on a 4-wheel drive RC model with a stereo camera and a lidar as the sensors. The setup along with some modifications with regard to the result of trials can realize the purpose of outdoor autonomous navigation under a controlled complex environment. After the tests, defects, like detection failure of lower obstacles or odometry drifting, were found. In future work, the reinforcement of odometry and the direct use of a human-based map on navigation can be carried out. As a result, our proposed robot with a promising autonomous driving capability looks forward to becoming a foundation for further research or application with self-driving cars on campus.

VI. Reference

[1] Taketomi, T., Uchiyama, H. & Ikeda, S. Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ T Comput Vis Appl* 9, 16 (2017).

[2] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

[3] Labbé, Mathieu, and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation." *Journal of Field Robotics* 36.2 (2019): 416-446.

[4] T. Qin, P. Li and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," in IEEE Transactions on Robotics, vol. 34, no. 4, pp. 1004-1020, Aug. 2018, doi: 10.1109/TRO.2018.2853729.

[5] http://wiki.ros.org/stepback_and_steerturn_recovery

[6]https://aaltodoc.aalto.fi/bitstream/handle/123456789/47114/master_Mets%C3%A4nen_Niilo_2020.pdf?sequence=1&isAllowed=y&fbclid=IwAR2Iq6-RXlUXSMN6WuuNhWYSe_hvT3PAeiNkbIWQQRlxnR7xLCOFIy1nag4

[7] http://wiki.ros.org/

VII. Project management and team cooperation

我們先從 SLAM 著手，完成建圖開始研究導航，接好導航包再用 Gazebo 再模擬的環境試跑，模擬環境成功後開始架設硬體和跑實體，在做每一個部分之前我們會先讀資料以了解整個架構，遇到問題就不斷上網查資料，我們每一個禮拜都會向教授報告本周進度，教授也會回饋給我們一些建議

團隊合作:

許晏誠: 建圖導航系統架設、Gazebo 環境架設與模擬測試、系統調整

何立平: 硬體架設、設備購買、硬體操作

張瀚: 導航、controller、硬體操作