An automatic food delivery robot based on ROS

Yintang Ni, Zeyong Zhang, Panagiotis Ntine, Yu Gan, Tianxiao Zhao

ABSTRACT

With various companies of every discipline trying to reduce their human work force in order to maximise profit, scientists and engineers are progressively considering concepts like utilising robots in order to perform simple tasks, such as hotel room service. In this paper we present a automatic restaurant delivery robot, as our final project. The project was conducted with the pioneer robot, model 3-AT, and the Robot Operating System (ROS). The robot is able to perform tasks in sequence, namely, pick up an order, plan a path to the corresponding table, while avoiding collision with walls and obstacles and identify the appropriate customer by using face recognition. The robot is expected to consider two alternative options: display a success message and return to the kitchen if the order was picked up by the customer, or return to the kitchen with the order, if no match was found within the time limit of one minute. The robot comes with a digital touch sensor, which determines whether the order was picked up or not, completely erasing the need for the user to provide input from the keyboard. It is fully capable of accomplishing its task in a realistic environment, since it can avoid moving obstacles, like people, as easily as it avoids static ones. Recovery from being kidnapped is not an issue, since it can easily reset its estimated position on the map and proceed to localise.

I. INTRODUCTION

The purpose of this report is to carefully and minutely describe the steps, design and experiments conducted during the actualisation of this project. The first section introduces the background and key technology implemented, followed by the design and implementation of concepts like autonomous navigation, face recognition and sensor detection, in subsequent subsections.In section 3, the robot model is set up and experiments are conducted in order to prove feasibility. In parallel, some improvements and existing problems are put forward in the discussion part. Finally, final thoughts and further work are included in the last section.

II. ROBOT SOFTWARE DESIGN

The purpose of the this project is to create a delivery robot that can be used in restaurants, delivering food from kitchen to aimed tables. The purpose basically requires the robot to own the capability of localization and navigation. In addition, some other functions which can make the robot more intelligent are better to be integrated in the robot, such as face recognition, voice broadcast and sensor detection for food. The main functions are shown in Fig.2 and each part will be discussed separately.

Fig. 1. line-tracking delivery robot

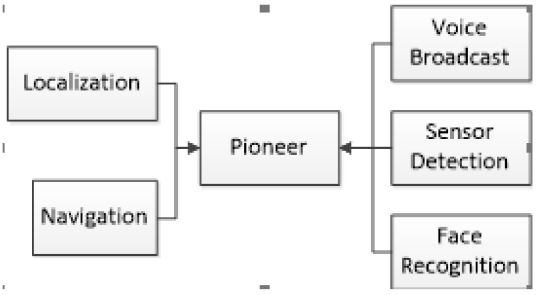

Fig. 2. Pioneer's functions

A. Navigation stack configuration

ROS provides the navigation stack (NavStack) for motion control, which contains ROS packages in path planning, localization, mapping, abnormal behaviour recovery and other aspects. Move_base is the core package and we can run this node for navigation. The input topics for move_base are /tf, /odom, /scan or /pointclouds, /map and /move_base_simple/goal, and the output topic is /cmd_vel, which is the planned velocity.

When the aforementioned move_base package is used, some if its parameters have to be configured. During the configuration process, the importance of some parameters became very clear. Before the move_base can be used, four files need to be created beforehand. Those files are base_local_planner_params.yaml, costmap_common_params.yaml, global_costmap_params.yaml and local_costmap_params.yaml.

In the base_local_planner_params.yaml file, 'pdist_scale' and 'gdist_scale' are extremely important, since those are

the two parameters that influence the robot's path on the map. In order to avoid obstacles in the local map, the value of 'gdist_scale' must be greater than 'pdist_scale', but it cannot be too great either, or the local path will depart from the global path, when the robot meets an obstacle. After experimentation with numbers between 0.1 and 5, a 'gdist_scale' value of 0.6 and a 'pdist_scale' value of 0.5 lead to a satisfying performance. Initially, when the robot reached the goal position, it started twisting indefinitely, because its twisting speed was too great for the robot to be able to recognise the goal orientation. A good solution wassetting yaw_goal_tolerance: 0.2, xy_goal_tolerance: 0.2.The parameter 'min_in_place_vel_theta' seemed to be related to that issue as well. If it is too large, the robot easily misses the right direction. The value was set to 0.01, which solved the problem.

In the costmap_common_params.yaml file, 'inflation_radius' determines the distance between the robot and walls/obstacles. The parameter needs to be handled carefully, because if its value is too high, the robot can not find the path to the goal pose, upon detecting an obstacle in the aisle. The reason for that is that the robot is trying to keep a safe distance from the walls and obstacles, preventing it from moving anywhere near them. Since a large portion of the map appears to be blocked, the robot will not find the available path to the destination, causing it to abort the plan.

In the local_costmap_params.yaml files, it is important for the local map to detect the obstacles and send the laser data to the global map, so it is updated accordingly. The local map needs to be refreshed constantly. It is a dynamic sub-section of the global map, and its purpose is to make obstacles visible on the static the global map. That sub-section is represented as a moving square on the map and it is referred to as the "rolling window". The rolling window's dimensions were set to 6 meters * 6 meters. The laser will clear obstacles outside of the rolling window.

B. Face Recognition

The face_recognition source code was used for the implementation of face recognition, and a camera was installed for that purpose. The software introduced is opencv2, very popular in this subject area. In order for the performance of face recognition to be improved, a concept of Machine Learning, K-nearest neighbours (KNN) of sklearn, was used to train face data sets.

The first step was the training of the face dataset, so that the location of the face in the picture can be identified. For that purpose, the function face_locations of face_recognition was used. The second step was to add the face encoding for current image to the training set. After using KNN classifier trained the training set, the result was saved to a file (knn.clf), which saved the trouble additional training, since, it could be read from the disk.

On the third step, face_locations was used to find the face in the camera, and transform the picture of the face's location to the KNN classifier, which read the data from knn.clf, and recognized the face. The success rate and error rate was controlled by configuring the distance_threshold, which is the recognition accuracy. If its value is too high, the recognition fails, by mistaking a person for another. After some experimentation, the value was set to 0.3, and the accuracy was excellent.

As a fourth and final step, the robot is commanded to return the order back to the kitchen if, after a minute of searching, it has not located the customer who placed it.

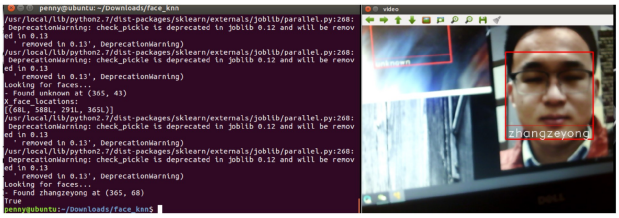

Fig. 3. Face Recognition

C. Sensor Detection

The pioneer robot is not equipped with arms in order to pick up the delivery, so it will not be able to realise whether it is carrying something or not. We improvised by adding a sensor. A digital touch sensor with two states was used in this situation. State 1 (True) means that the delivery is on the platform, and state 2 (False) means there is nothing on the platform, so the delivery has been taken away.

Another obstacle was the fact that the sensor cannot connect to the laptop directly, since there is no USB interfaces, so it needs a microprocessor as a bridge. The microprocessor collects sensor data and convey them to the laptop via serial ports. Arduino is used as a microprocessor in this robot. There are two ways to realize data transformation. The first one is to use ROS serial library, realizing data exchange directly and the second one is to use IDE in ROS, making it work as a node. The second method was used in our robot. For the Arduino part, a publisher is initialized to publish the touch sensor's boolen state data that was collected by the digital input port of Arduino. For ROS, a subscriber is initialized to subscribe to the node, using a 'for loop' to search 10 times for the existence of an object (in this case, the food to be delivered). If the outcome is True for all 10 repetitions, it means that the delivery has not been taken away, causing the state machine to turn to the aborted state. If the 10 times result returns False values, it means that the food delivery task is finished and it can turn to the success state.

The sensor detection part solves the problem that customers need to use keyboard to confirm the food has been delivered successfully, improving the degree of automation. For improvement, it would be better to add other sensors to realize human-computer interface, or use the camera to realize object recognition.

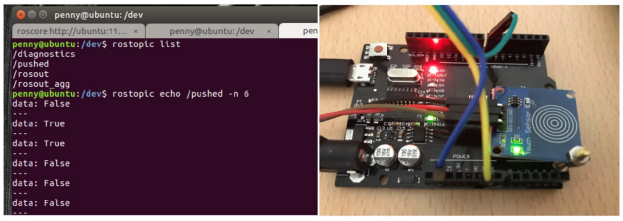

Fig. 4. Sensor Detection

D. Smach State Machine

Smach state machine contains finite states and each state corresponds to a robot action. When the current state finishes, the state machine will jump into the next state, which is decided by the executing outcome of current state. Using state machine makes the robot realize task-level control, which is convenient to control the robot's complex behaviour.

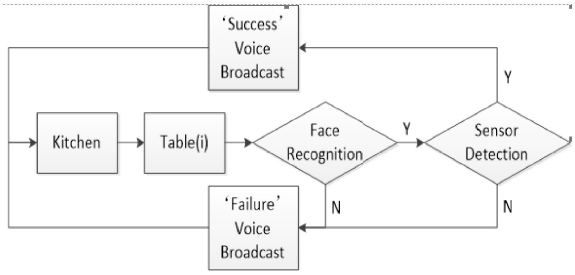

Fig. 5. State Machine Overview

Figure 5 shows the overview of the machine that we used in this robot. Firstly, the robot is given the goal table number as an input from the keyboard. Secondly, the robot uses navigation stack to go to the kitchen, take the order and then travel to the appropriate table. Lastly, the robot will make some judgements: face recognition for finding the correct customer and sensor detection for checking whether the food has been taken away. If the two judgements both show True, it means the robot successfully finishes the delivery task, it will broadcast a 'success' message. Otherwise, it will consider the delivery is failed and broadcast a 'failure' message. Finally, the robot will go back to the kitchen for the next delivery.

The smach state machine part improves the stability of the robot, which means that developers can check the running condition of the robot with debugging and operating data. Once more functions are integrated to the robot, the state machine would show its advantages.

E. Solution to robot kidnapping problem

Since our initial solution to the kidnapping problem was not ideal, many improvements were made. Initially, the goal position is identified. If the robots gets kidnapped, the weight from the laser will be reduced. After testing the weight, we find that when the max_weight ¡ 5, it means the robot is kidnapped. As a result, he function 'replace particlecloud' will be called, which will randomly spread 1000 particles to the map, and when the robot moves, it's position will be reset.

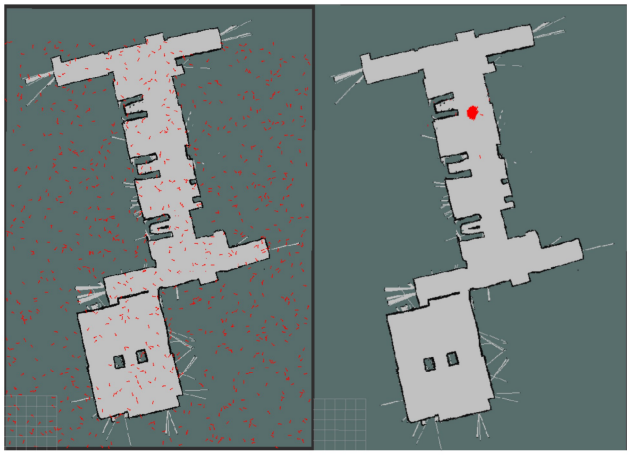

Fig. 6. Put particles random in the map and reset the position of robot

F. TF Transform

Several problems were met regarding tf transform. In our second lab assignment we found that when we used the pf node, rviz always displayed an error in the frame that there is 'no map'. We later realised that this happened because the there was no tf transform between map and odom, so we used rosrun tf static_transform_publisher 0 0 0 0 0 0 1 /map /odom 10. However, in this lab assignment, the static transform had many issues. First of all, the move_base needs the odometry to match the robot's actual position, since the move_base needs the robot's current position in order to plan the global and local paths. Additionally, since move_base used odom, and odom position needs to be the same as the robot's current position, the static_transform_publisher was used. The odometry could not match the position of robot. In order to fix this problem, a new publisher was created in the pf node. When pf node is run, it will initialize a particlecloud in the first position(0,0,0), so it will fix the tf problem between map and odom. We can see in the following picture that the blue arrow of odometry is same with the estimate position of robot.

III. Experiment Results

After realizing and debugging the software designs above, we finally made the food delivery robot. As a first step, the robot operator needs to send a initial pose to the robot, and then input the aimed table number, the robot will navigate to the kitchen and take food. After that, the robot will navigate to the aimed table, avoiding obstacles during the process. The robot will check certain customers and whether the order has been taken away via face recognition and sensor detection. If the return values are both True, it means that the food delivery task has been accomplished and the robot will generate a 'success' voice broadcast. Otherwise, the robot will generate a 'fail' voice broadcast. Finally, the robot will go back to the kitchen and wait for next task. The robot has a good performance in general, realizing all functions, but there are still some improvements can be adapted. When the robot updates obstacles from local costmap to global costmap, the obstacles exist in the global costmap for a long time and can

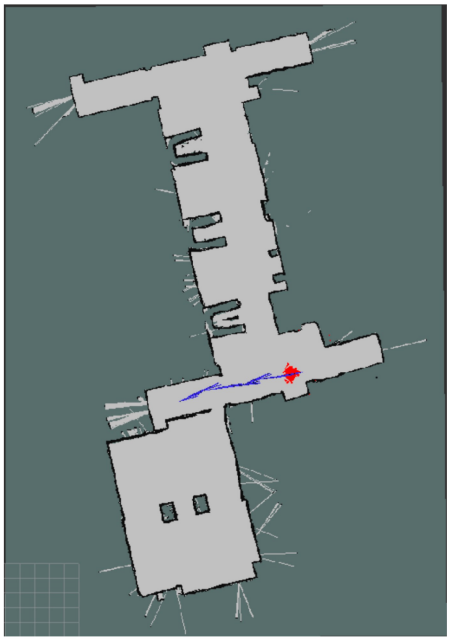

Fig. 7. TF between odom and estimate position

not be removed easily, even though the robot turns several rounds. It often happens in narrow space and causes the robot to get stuck there and, consequently, quit the path plan. Finally, if we make the robot revolve when conducting face recognition, it would be more intelligent and user-friendly.

Fig. 8. food delivery robot presentation

IV. Discussions

For VMware, when external USB devices are connected to the virtual machine, it is necessary to connect them manually via the removable devices settings. It also necessary to choose the USB2.0 and USB3.0 compatibility setting.

When using pyttsx module in state machine for voice broadcast, we put object initialization and parameter settings into '__init__' function and put voice speak sentences into 'execute' function. It doesn't work, which means that the robot only broadcast a few words and then return outcome and then jump to the next state even with the runAndWait() function. After that, we put all sentences into a function and then call this function in 'execute' function and it works. When calling functions, the program will waiting for the ending of the program, even without return values.

When running pioneer in our own computer, we need to install drivers for both laser and robot motors. The solution is to copy the 'socspioneer' folder to our workspace and use 'catkin_make' command to rebuild the workspace. When lacking related packages, we can use 'sudo apt-get install ros-kinetic-packagename' to install these packages.

When using the map we created with gmapping package, we find that noise in the map infect the navigation performance to a large extent. We solve the problem after using Photoshop software to remove these noise. Noise is inevitable, but if we control the robot at a low and constant speed when building the map, it would be better.

Fig. 9. Original map picture and Processed map picture

V. Conclusions and Further Work

The key point for this kind of robot is to realize localization and navigation. Many existing robots mainly use photoelectric or electromagnetic sensors to detect the settled lines in restaurants, which is short of avoiding obstacles during line tracking process. If we use laser radar for localization and navigation, it would have a better performance. Furthermore, if some other functions for task check and people-oriented service are contained in the robot, it would be better for popularization.