Knock Knock Robot using Pioneer & Kinect

Sing-Yao Wu, Hung-Hao Chen, Vitor, Yun-Hua Wang
[1]Department of Computer Science and Information Engineering, National Taiwan University

***Abstract*— This paper addresses a method to visit all the doors in the same floor with a mobile robot, from door-to-door in order to send speech messages during festivals automatically. In order to do so we use a Pioneer combined with a Microsoft Kinect and a laser device. We select an operating system that is flexible to integrate several systems together. We also build a local environment map by laser-based mapping system and localize the robot by a probabilistic localization system. Moreover, we use a fast interpolated navigation function to plan a reachable path for each door and a face detection method so as to determine if someone comes out from the visiting door. This work can send speech messages between several designated locations with reachable paths if there is a person at those particular locations.**
Keywords: SLAM; Localization; Navigation; Face detection.

I. INTRODUCTION

Visiting neighbors, friends and relatives to send New Year wishes is one of the important tradition in Taiwan. However, people are tend to send New Year wishes via multimedia such as chat applications and voice messages on phone nowadays. Although it is a convenience way to send all the New Year wishes in few steps or even in one click, it cannot fulfill the meaning of this tradition without greeting at the door. On the other side, robots are becoming a part of our life following the growth of hardware and software improvements. Techniques and developments on mobile robots are adequate on dairy use. Therefore, we address a method to visit all the doors in the same floor with a mobile robot, from door-to-door and send New Year wishes to the person who answered the door (Fig. 1).

In this work, we tackle problems on localization and navigation on a pre-generated map using Pioneer 3-DX, which is the main goal to visit every families that nearby. Also we introduce SLAM system, which uses a laser device to retrieve environment data to build a local map. Moreover, we implement face detection in order to determine if there is a person answered the door.

II. RELATED WORK

As far as we know, there haven't been similar works proposed by other authors. We decide to split our work into three major parts and discuss their related work.

A. Simultaneous localization and mapping

Simultaneous localization and mapping (SLAM) has been an important area in the robotics for decades [1]. SLAM is also becoming an increasingly necessary topic in computer vision community, receiving particular interest from not only virtual reality (VR) but also augmented reality (AR) industries. With the help of SLAM, robots can easily and accurately create a map of its surroundings and locate current position in the map. A simple tutorial showed that the simulated robot is able to build the map of the nearby environment through SLAM by letting the robot move around [2]. Not only in the simulated environment, people also used SLAM in the real world. The students in MARHES lab at University of New Mexico demonstrated that how Pioneer 3-AT used SLAM to build a map of a handcrafted environment with some obstacles and slopes, while simultaneously getting the pose information of Pioneer 3-AT [3].

Fig. 1. (Left.) Pioneer visits door. (Right.) A real-life situation of visiting doors.

B. Navigation

The ability to navigate in the environment is one of the most important capabilities for mobile robots [4]. Navigation includes 2 tasks: First, guiding the robot to a specific location in the environment.Second, avoiding some dangerous situations, for example: collision, and staying with safe operating conditions, such as: high temperature, radiation, exposure to weather, etc. Robot navigation means the ability to determine its own position in the map and plan a path towards the target goal point afterwards. The students in MARHES lab showed that navigation with Pioneer 3-AT can be implemented so as to have the mobile robot go to a designated location [3]. Besides simply navigating the robot to particular locations,

[1]Devices are supported by Intelligent Robot and Automation Lab, National Taiwan University.

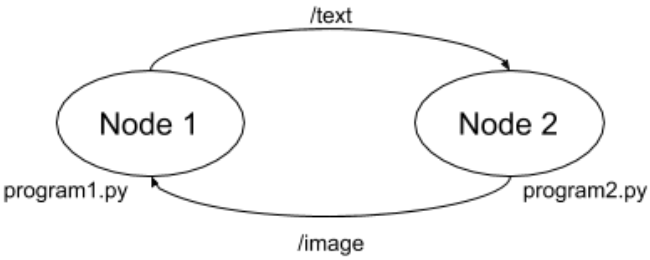

Fig. 2. Illustration of two different programs communicate with each other through the topic in ROS.

Stanley Robotics came up with an idea to integrate the navigation with the empty parking spaces searching [5]. This robot called "new Stan", which can help customers to search for parking spaces automatically while the customers only need to park the car in a designated place.

C. Face detection

Face detection is an important aspect of effective human-robot interaction (HRI). With the rapid development of deep learning, the accuracy of face detection has stepped one big step forward and a lot of methods have been proposed for implementing face detection. Li et al. [6] proposed a cascade architecture built on convolutional neural networks (CNNs) with very powerful discriminative capability, while maintaining high performance. Zhang et al. [7] presented a real-time face detector, which performs superiorly on various scales of faces with a single deep neural network, especially for small faces. YOLO [8] is a state-of-the-art real-time object detection system, which uses neural networks as the fundamental model. It is extremely fast and accurate.

III. METHODOLOGY

In this section, we describe the Robot Operating System (ROS) and some related packages we adopted to implement SLAM and navigation in detail. Then, we introduce the procedure we use to carry out face detection with the help of Kinect.

A. ROS

ROS [9] is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. In ROS, each program is considered to be a node, which can communicate with each other (Fig. 2). Nodes can publish/subscribe a specific topic to send/receive data. With the help of ROS, we can efficiently let multiple programs communicate with each other so as to manipulate the robot. The followings are the related ROS packages that we use to implement SLAM and navigation:

1) gmapping: Gmapping [10] is a ROS package providing laser-based SLAM. Using gmapping, we can create a 2-D occupancy grid map (like a building floorplan) from a laser device of a mobile robot and also collect the pose data of that robot (Fig. 3). As long as the map is created, we can save the map and load this map for future usage, so that we need not run gmapping again.

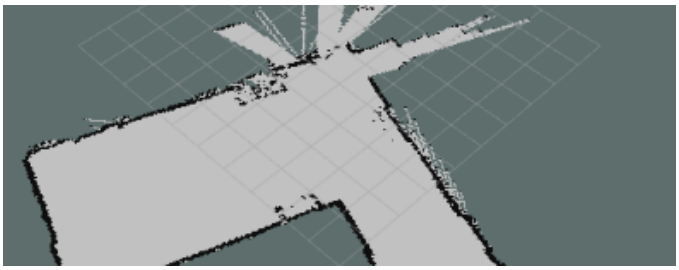

Fig. 3. The map around built through ROS gmapping.

2) Adaptive Monte Carlo Localization (AMCL): After building the map with gmapping, we need to let the robot know its position in the map. AMCL [11] is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. Once AMCL is running on the computer, it collects data from the laser device and the map so as to locate itself on the map as the mobile robot continues to move. As the time goes by, the pose of the robot will eventually converge to nearly one point (Fig. 4).

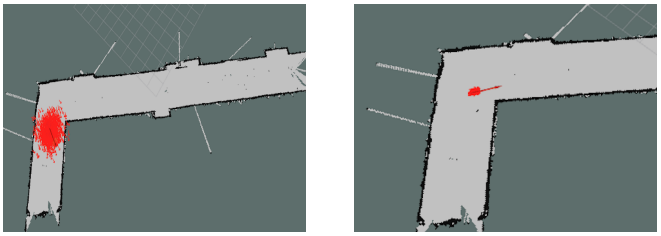

Fig. 4. The process of convergence of the pose of the robot.

3) move_base: This package [12] is used to provide an implementation of an action that, given a goal in the world to a robot, it will attempt to reach the goal with a mobile base. The move_base node links a global and local planner together to accomplish its global navigation task. If we send the target goal point to the topic that move_base subscribes, the move_base node will start to plan a path to the target based on the given map and the received laser data. In our experiment, we adopted NavfnROS as the base global planner and Dynamic Window Approach (DWA), which is more efficient, as the base local planner.

B. Face detection

Before talking about face detection, we have to elaborate on the Microsoft Kinect, which is a device we used to capture images. Kinect is a line of motion sensing input devices that was produced by Microsoft. By means of Kinect, it can be the sight of the mobile robot, which makes the robot able to collect vision data from its facing direction. In order to collect the image data from Kinect, we have to install a ROS package called **freenect** to extract image data [14]. We run the launch file of freenect package so as to publish the

received images from Kinect to a topic. Other programs only need to subscribe that designated topic to get the image data.

As we mentioned above, many neural networks approaches have been proposed to implement face detection, many of which showed competitive performance. Due to limited computing resources, we choose to use the histogram of oriented gradients (HOG) model [13] to detect faces in an image. HOG is a feature descriptor used in computer vision and image processing for the purpose of object detection. This technique counts the occurrences of gradient orientation in localized portions of an image. After the face is located, we draw a bounding box around the face which makes it more clear in an input image (Fig. 5).

IV. EXPERIMENTAL SETUP

The following presents two parts of our experiment: simulation, and hardware experiment. We use Robot Operating System (ROS) to implement our algorithm. Aforementioned, ROS is a set of sortware libraries and tools that makes robot applications building become easier. We send the computation result to the robot by publish the result to a ROS topic, which is a common method to communicate between nodes, or processes in ROS. Using this method, we can easily control a virtual robot in simulator or hardware robot without changing the structure of the system we designed.

A. Simulation

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to conveniently test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios without the risk to damage the hardware. At first, we use Mobilesim toolkit to simulate the algorithm we designed. After testing, we found that Mobilesim seems to be too simple for implementing our algorithm on it.Then we turn to use Gazebo toolkit to simulate the algorithm we designed. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments [15].

Fig. 5. The result of face detection.

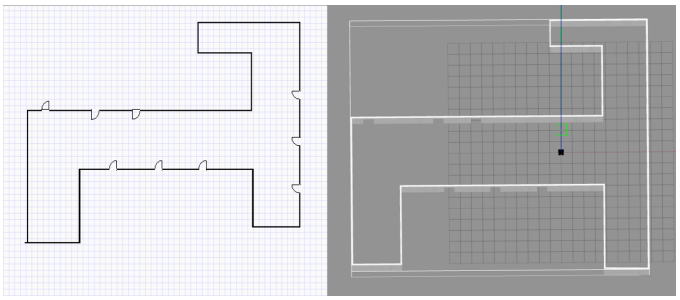

Fig. 6. A model of second floor of Ming-Da Building using the Building Editor in Gazebo Simulator.

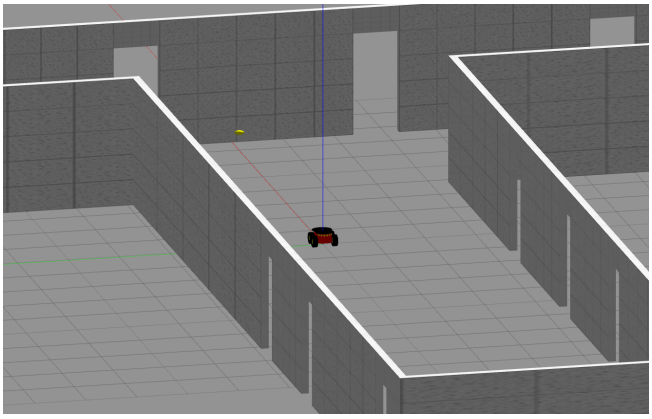

Fig. 7. Simulation in Gazebo. A virtual Pioneer 3-AT is running in second floor of Ming-Da Building (model).

We use a virtual Pioneer 3-DX as the mobile robot to test our algorithm, and use a virtual Pioneer 3-AT to validate whether our algorithm can be implement on other robot or not. Then we create the environment for the virtual robot to run on. We build a virtual second floor of Ming-Da Building of National Taiwan University using the Building Editor in Gazebo Simulator. Fig. 6 shows the result. Finally, we connect the Gazebo simulator with the other ROS system so that the virtual robot can get the command from other part of the system. . Fig. 7 shows how it works on Gazebo.

The most difficult part of the simulation is that some of the physical phenomenon isn't be simulated in the simulator. For example, teleop_twist_keyboard is a package which can control robot using keyboard. It works well on a real Pioneer 3-DX: robot runs when a corresponding key is pressed, and stops when keys are released. However, when we use it on a virtual robot on Gazebo, after a key is pressed, virtual robot will keep moving even if keys are already released. We conclude that it is because Gazebo doesn't simulate friction on a virtual robot, so that it will keep moving even when it hits the wall.

B. Hardware Experiment

We then implement our algorithm on a real Pioneer 3-DX. The operating system of the computer is ubuntu 16.04 LTS with ROS kinetics. We use Kinect 2.0 to implement face detection. For the laser, we tried two kinds of laser devices for

	SICK LMS100	Hokuyo URG-04LX
max scanning range	20m	5.6m
max scanning angle	270°	240°
precision	30mm	3% of the distance
input voltage	10.8V - 30V	5V

TABLE I
DIFFERENCE BETWEEN SICK LMS100 AND HOKUYO URG-04LX

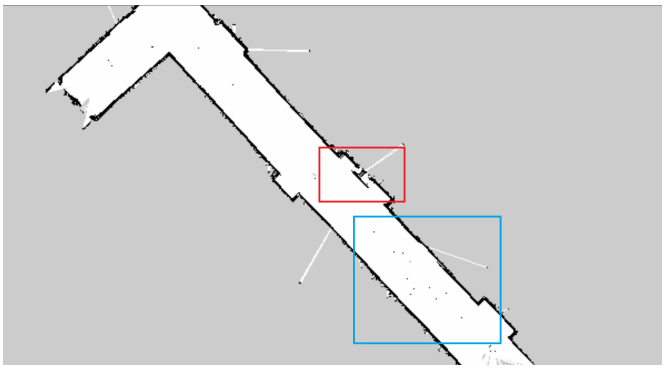

Fig. 8. Problems happened when using laser. The red rectangle shows that a wall is misplaced between two door. This may result from low accuracy, too short range of the laser, or error between velocity command and real velocity of the robot. The blue rectangle shows some black points that doesn't exist in the real world is drawn on the map. This may result from low accuracy of lasers, or human detected since staying too close to the range of laser.

SLAM and navigation: SICK LMS100 and Hokuyo URG-04LX, both have different characteristic. SICK LMS100 has larger range so that it can detect longer distance than Hokuyo URG-04LX, but consume more power. Therefore, it needs to connect to a battery with high voltage to offer the power when using SICK LMS100. In the contrast, Hokuyo URG-04LX can be charged by just connecting to the USB of the computer, while its range is much more shorter than SICK LMS100. The detail comparison is shown in Table I .

We noticed that when using Hokuyo URG-04LX for SLAM to construct the map, SLAM sometimes draws same objects on different locations of the constructed map, this problem may result from low accuracy of lasers and shorter range of Hokuyo URG-04LX, or error between velocity command, real velocity of Pioneer, and velocity detected from sensor. Another problem that happens when constructing map is that, SLAM sometimes draw black points on empty location to the map. This happens on both of the laser devices. The problem may result from accuracy of the laser devices, or human staying too close to the range of the laser, caused detected. The two problem are shown in Fig. 8. The solution we conceives to reduce the error is that use SICK LMS100 to construct the map, then clear the black points manually using image editor like **gimp** on Linux. After finished these steps, switch the laser to Hokuyo URG-04LX for the navigation, which is more convenience to charge.

V. RESULTS

We divide the results of the experiment into three parts. The first two parts explain how we tune the parameters of the navigation algorithm to get the most suitable parameters for Pioneer. The third part shows the final result after we implement our algorithm on Pioneer 3-DX.

A. Velocity Parameters Tuning

For the navigation part, we compare two kinds of navigation methods provided by ROS: Dynamic Window Approach (DWA) Planner and Trajectory Planner. After testing, we found that Trajectory Planner may make Pioneer run too fast, and sometimes Pioneer doesn't follow the path Trajectory Planner computed. It is dangerous to use this planner because it will put our robot in danger to hit obstacles. As a result, we use DWA planner to implement navigation.

There are a lot of parameters need to be adjust to fit our robot. The definitions can be found in [16]. The first important parameter is the max and min x velocity and y velocity, which means the velocity in the direction parallel to robots straight movement and the velocity in the direction perpendicular to that straight movement. The default max x velocity is 0.3 and y 0.0. When the testing uses default value, the robot will move too slow that turns out the robot move only one wheel each time. We guess that the reason is the max velocity is too low and lack of y velocity, which lead the robot moving with too many constraints. Therefore, we tune the max x velocity to 1.0 and max y velocity to 1.0. Similarly, the limit of acceleration in X and Y direction is tuned from 0.3 and 0.0 to 1.0 and 1.0. The max rotational velocity of the robot is tuned from 0.5 to 1.0, and min rotational velocity is tuned from 0.2 to -1.0, which may let the robot able to turn both clockwise and counter-clockwise. Also, the limit acceleration of rotation is increased from 0.1 to 0.5. The testing result and demo video shows that these velocity-related parameters can make Pioneer move smoothly.

B. Trajectory Parameters tuning

After the path to goal is calculated by the DWA planner, the trajectory of the robot (i.e. how is the robot going to move) is determined by the cost function

$$\begin{aligned} cost = &\text{ path\_distance\_bias} \times dist_{pt} \\ &+ \text{goal\_distance\_bias} \times dist_{gt} \\ &+ \text{occdist\_scale} \times cost_{obs} \end{aligned} \tag{1}$$

where $dist_{pt}$ is the distance to path from the endpoint of trajectory, $dist_{gt}$ is the distance to local goal from the endpoint of trajectory, $cost_{obs}$ is the maximum obstacle cost along the trajectory in obstacle cost (0-254).

The objective is to minimize the cost function. There are three parameters we can adjust: path_distance_bias, goal_distance_bias and occdist_scale. Path_distance_bias is the weight for how much the planner should stay close to the computed path, which means that increase this parameter will make planner prefer to follow the trajectory of the computed path. Goal_distance_bias is the weight for how

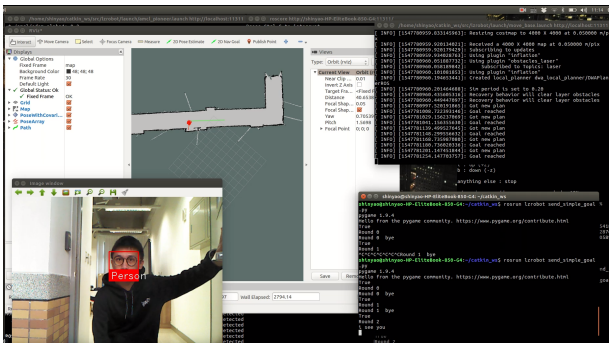

Fig. 9. A snapshot that shows how the system interface looks like in the screen.

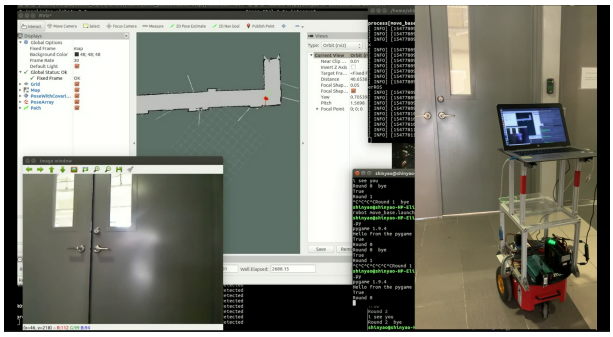

Fig. 10. The demo shows how the work looks like in a pre-constructed map.

much the robot should attempt to reach the local goal. Experiments show that increasing goal_distance_bias enables the robot to be less attached to the global path, shown in [16]. Occdist_scale is the weight for how much the robot should attempt to avoid obstacles. A high value for this parameter results in high probability for a robot to stuck in place while moving.

We found that sometimes Pioneer doesn't follow the global path too close and thus approaches too close to the wall when passing a corner, so we increase path_distance_bias from 32.0 to 50.0 to make the robot stay closer to the global path. Also, we increase occdist_scale from 0.01 to 0.02 to avoid the robot to hit obstacles and wall. But after testing, the improvement of the performance isn't very obvious, and sometimes makes the cost become a negative number. We choose to use the default values in the end.

C. Hardware Experiment Result

We connect Pioneer 3-DX to our computer using RosAria package, connecting Hokuyo URG-04LX laser to our computer using urg_node package and connecting Kinect 2.0 to our computer using freenect package. After all devices are connected to the robot, our ***Knock Knock Robot*** is formed. We localize the robot using teleop_twist_keyboard package and amcl_pioneer.lauch, let robot is able to navigate using move_base.launch. Additionally, we write our own node to become a main controller of the robot. It determines where are the doors and send the goal points to move_base.launch to execute navigation. After the robot achieve a door, it simulates the knocking action by making some noise that sounds like knocking a door/. Then Kinect will detect if any person comes out from the door by face detection. The interfaces in the screen are shown in Fig. 9. If it detects any human faces, it will send message to the main controller. The main controller then plays a sound recording saying "Happy New Year" or other sentences. The robot will keep visiting from door-to-door until all doors setting in the program has been visited. A hardware demonstration results are shown in Fig. 10 and a demonstration video can be found at here: https://youtu.be/5CdkwZ7k85g

VI. CONCLUSIONS

This work presents ***Knock Knock Robot***, a robot that can visit from one door to another door, knock the door and say some speech messages to the person who opened the door. Techniques and applications: SLAM, ROS, face detection, for instance, are implemented to the robot. We did experiments including simulation with Gazebo, hardware experiment with Pioneer 3-DX, and we found and solved the problems such as error on scanned SLAM map due to inaccuracy of laser, control problem during navigating due to inappropriate parameters in DWA planner, and other technical problems.

At this stage, out robot doesn't have the ability to recognize the door and move to the door simultaneously. And also, the robot can only say the pre-recording sentence and cannot talk to people spontaneously. The future work will be solving these problems by implementing real-time object detection package like YOLO, and chatbot to talk to people. Another possible future direction is to add facial expression recognition and let the robot to change its word according the facial expression of people. We believe this kind of social robot will be the trend in the future.

REFERENCES

[1] An introduction to SLAM: https://goo.gl/auMtqj
[2] SLAM tutorial in a simulated environment: https://goo.gl/4mZAnA
[3] SLAM and navigation using Pioneer 3-AT: https://goo.gl/9rRu4h
[4] A introduction to navigation: https://goo.gl/1m3rzs
[5] Stanley Robotics, "Meet the new Stan, the first outdoor valet parking robot!": https://goo.gl/17BXkf
[6] H. Li, Z. Lin, X. Shen, Jonathan Brandt and Gang Hua, "A Convolutional Neural Network Cascade for Face Detection," *in CVPR*, 2015.
[7] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang and Stan Z. Li. "S3FD: Single Shot Scale-invariant Face Detector," *in ICCV*, 2017.
[8] You only look once (YOLO): https://goo.gl/hVqq8c
[9] Ros official: https://goo.gl/E1tRJ5
[10] An introduction to ROS gmapping package: https://goo.gl/GyLA3C
[11] Adaptive Monte Carlo localization: https://goo.gl/SQKRi4
[12] How to implement path planner in ROS: https://goo.gl/ovVFKf
[13] Histogram of oriented gradients model: https://goo.gl/bCi5Sx
[14] Freenect ROS wiki: https://goo.gl/aHcvmH
[15] The offical website of gazebo: http://gazebosim.org/
[16] Kaiyu Zheng ROS Navigation Tuning Guide: https://goo.gl/km35JV