Mobile Robot Platform at Nursing Home for Elderly with Application Service

Choi JiHwon, Jeong JongHyeon, Kim JunSeo and Lim SeungHyeon

Capstone Design(SWE3028), SungKyunKwan University

Abstract. Recently, as the average population ages has been growing rapidly, the need for services for the elderly is increasing. However, compared to the increasing demand, the working environment of employees who take care of the elderly is getting worse. To solve this problem, we propose a mobile Robot platform that can manage the elderly with an application service. The platform consists of three main functions : Robot call on user request, Emergency detection for fall-off motion, and stroll assistant. We conducted experiments on these three functions on indoor and outdoor maps in the gazebo simulator. The robot in a simulated environment successfully follows the expected scenarios and shows good performance in the experiment. Although there are limitations in fall-off detector and tracking algorithms, there is sufficient potential to further develop in not only the simulation but also the real environment through functions such as face recognition and GPS.

Keywords: Robotics · Mobile app · Simulation · Pose estimation · Sort track

1 Introduction

Korea is rapidly turning into an aging society due to a decrease in fertility rates and an increase in life expectancy. Accordingly, nursing homes have become common and the demand for nursing care workers has increased, but they are under poor working conditions. According to the survey on nursing care workers, 27.5% of workers at nursing home have a experience of getting damage by others , and 45% are dissatisfied with their working conditions. The reasons for the dissatisfaction are the low salary, the attitude toward clients, job insecurity, and difficulty of work[1]. Accordingly, in this Capstone project, we tried to solve the problem by developing a robot service that can be used in a nursing home. We supported three functions of the service we want to develop. First, we developed a function of calling the robot to a location desired by the user. Second, the robot gives an alarm to the caregiver in the event of an emergency such as a fall accident. Finally, it assists users to take a walk with them. Through the service that supports the above functions, it is designed to reduce the labor of nursing care workers to increase work satisfaction, and to allow users to live a safe and pleasant life without feeling uncomfortable using it.

2 Backgrounds & Related works

2.1 Object detection

There have been many techniques for classifying images into pre-trained objects through deep learning models. In addition, as the image processing speed has dramatically developed, object detection in moving images as well as still images become possible.

Object detection includes a two-stage method that provides selective navigation through area suggestions, and a single-stage method that calculates the probability that each area contains an object by dividing the image into predefined grid areas.

First, the two stage method has a model of the Region-based Convolution Neural Network (R-CNN) series. The R-CNN-based model has the advantage of being generally more accurate than the single-stage method, but has the disadvantage of being slow in image processing because CNN learning is performed after setting a section with a high probability of having an object as a Region-of-interest (ROI).

On the other hand, You Only Look Once (YOLO)[2] is single-stage method. The YOLO model divides the input image into grids and obtains bounding boxes and confidence through grid cells, indicating where the object is located and the probability that the object is contained.

In this project, YOLO was selected for application in a real-time environment. Object detector is used to enable robots to recognize people and track their location. In addition, it is easy to expand into various services such as object transport and collection through object learning in future studies.

COCO dataset was used for anonymous human detection in the simulator, and it was well applied to the anonymous model in simulated environment without additional learning.

2.2 ROS(Robot Operating System)

The robot consists of numerous sensors and devices, and accordingly, the software is designed in a complex manner. Therefore, in order to control the robot, it is necessary to build a system that integrates and manages all of thes . In this project, the overall framework was constructed using the Robot Operating System (ROS) based on the TCP network. The ROS consists of several nodes. Each of these nodes exchanges messages with each other to exchange data. By dividing one huge robot framework into nodes, the program can be divided and be able to collaborate with other people. In addition, ROS eliminates problems related to data standardization and API compatibility that may occur due to segmentation.

Figure 1 shows a simple ROS network through topic. All nodes are always connected to *ROS MASTER*. The image data received from the camera goes to *Camera Node1* first. The *Camera Node 1* node simultaneously publishes the topic */image_ data* to *Image Processing Node* and *Image Display Node*. In this

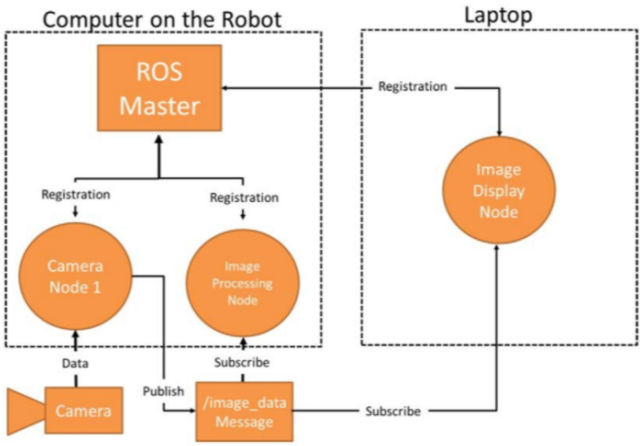


Fig. 1. Simple ROS network exmaple

way, the image is displayed on the laptop through *Image Display Node*, and *Image Processing Node* processes raw images to better understanding of it. In this way, the ROS may transmit data through a node and implement various functions.

In this project, sensor data of the robot is loaded through ROS, and the robot is controlled by ROS also. In addition, the Navigation Stack provided in the ROS open-source package allows the robot to move without additional mapping or localization. Finally, ROS is well linked with the gazebo simulator and openv, an image processing tool, so it was possible to easily connect the simulator and robot. Object detection and navigation through image processing were also possible.

2.3 Gazebo Simulator

In this project, the goal was to implement robot in the simulator, a virtual environment, to avoid the financial and spatial constraints of robot production. Gazebo simulators are 3D dynamic simulators and are generally used to test robot algorithms or scenarios. Gazebo has high capability of connecting with ROS and provides a unified interface for each API. In Gazebo, the test can be conducted using various sensors such as LiDAR sensor and camera sensor, so there are few restrictions on robot manufacturing. Since Gazebo is an open source with an Apache 2.0 license, there are also few restrictions on development.

2.4 Pose estimation

Pose estimation is a computer vision technology that detects a person's posture. It is a common technique that detects the location and direction of an object in computer vision, measuring and estimating the location of a person's body

joint, the keypoint, and how it is organized. Pose estimation has two approaches: top-down and bottom-up. The top-down method is a method of detecting a person first and then estimating each person's posture, and there is a disadvantage in that it cannot be measured if a person is not recognized, and as the number of people increases, the amount of calculation also increases. On the other hand, the bottom-up method detects joint areas first and connects them to each other to estimate everyone's posture, but there are many combinations that can match the joints found, and it takes a long time to match properly, so there is a problem of accuracy.

A. Realtime Multi-person 2D pose Estimation - CVPR [3]

This is a method of learning a filter capable of encoding a limb (a joint-tailored portion of a body part) containing location information and direction information of a body part as a 2D vector. The accuracy and speed of pose estimation were improved using a bottom-up approach that performs pose estimation by several people.

B. OpenPose: Realtime Multi-Person 2D Pose Estimation - TPAMI [4]

The TPAMI version finds Part Affinity Fields (PAFs) and Part Confidence Maps, which are heat maps containing information on the location of joints and bones. While connecting the two models in series, they were able to improve the model in terms of memory and speed. Confidence maps that inform the location of joints can be found accurately and quickly through PAFs.

PAFs can be trained through ground truth data. Therefore, pose estimation is performed by connecting the joints found through PAFs. In this model, redundant PAF connection was added to increase accuracy for nearby joints.

2.5 Navigation

The robot's driving function uses the navigation package of the ROS. For navigation, you need to know the current location of the robot on the map through localization and receive the destination as an input value. Based on this, the global route is calculated, and the robot keeps updating the location of itself as it moves, continuously modifying the route to the destination in consideration of surrounding obstacles. Therfore robot finally arrives at the destination. We are applying the LiDAR geometry method using LiDAR sensor to the odometry that identifies the robot's current location. It compares map and sensor information and tracks the robot's current location.

2.6 Sort track (Simple Online Real-time Tracker)

When processing an image, problems may occur when there are multiple objects rather than a single object. SORT track is a tracking algorithm that is

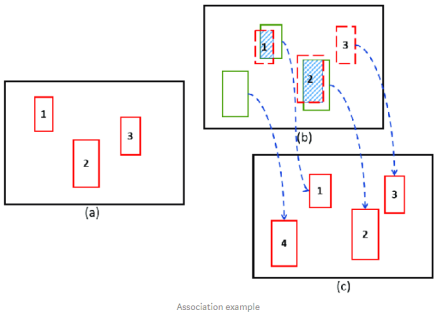


Fig. 2. Bounding boxes matching by Hungarian Algorithm

simple in principle and lightweight in computation. It can be used with several object detection tools. The key to tracking is to effectively correlate real-time with bounding boxes of several objects. Simple Online and Realtime Tracking (SORT)[5] is most widely used among open source tracking algorithms, especially because it is simple and fast. The SORT track take bounding boxes of the object as an input, and goes through Kalman filter[6] and Hungarian algorithm stages[7]. First, Kalman filter predicts the possible future positions and velocities of objects. In other words, the set F of bounding boxes detected in the *nth* frame estimates F^* in the *n+1th* frame with a Kalman filter. Next, it compares the bounding boxes F* predicted by Hungarian Algorithm with the actual ones and matches them as shown in Figure 2.

2.7 Overall architecture

The overall structure is as Figure 3.It was implemented to process the user's input received from webpage (Front-End) at the server (Back-End) and then deliver it to the robot (Robotics).

Robotics Overall framework of robotics are based on ROS nodes. ROS framework controls the entire robot from sensors to actuators as well as other important functions such as navigation. We are able to send or receive data topics, especially from sensors by ROS with regularized data types. We are able to connect with server by *rosbridge* package. A robot sends a location of itself, and compressed camera images to a server, and server is able to order a robot where to move as well as other calls from applications.

Backends Back-End part of service divide into Server and Storage DB. The server operates on EC2 in AWS. Storage used AWS' S3 and Database used PostgresQL to operate on AWS's Lightmail. In order to effectively implement the server, components were implemented to connect each other by dividing it into five files as in below figure. This also made it easier to facilitate maintenance for each function.

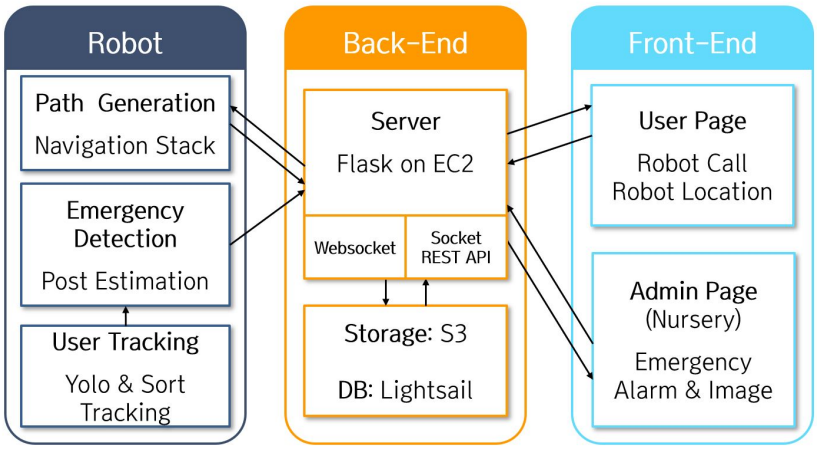


Fig. 3. Overall architecture of the Service

URL	Function
/register	It is an API that receives the user's personal information, stores it in the database, and completes the membership.
/login	It is an API that receives the user's ID and PW, distinguishes the user according to the database, and performs a login function.
/info	It is an API that returns the user's ID and type using a session.
/call_sebot	It is an API that receives an index of a room where a user can move on a map and allows the robot to move to the corresponding location.
/get_image_list	It is an API that informs all accident records (photographs and robot locations at the time of the accident) of the nursing home when the caregiver approaches the notification page.
/get_map	When the "Map View" button is pressed on the notification page, API uses the robot's location to locate the robot's location into the map and draws the location on the map as an administrator to grasp the location at once.
/robot_location	It is a socket event that draws the location of the robot on a map so that the user can know the location of the robot in real time.

Table 1. Flask Function in App.py

App.py App.py is the main script for running the server and reads the bottom four classes from the file. In addition, six REST APIs and one socket were

written to implement the sebot service. To implement this functionality, flask frameworks (flask, flask_socketio, flask_cors) were used, and other OpenCV and base64 libraries were used to modify maps. Each component has the following functions.

Config.py Config.py is a file containing information about Storage and Database. For convenient, the information was put on file, but in reality, it is planned to be stored in a secure location such as /mnt/secrets.

db_ utils.py In order for Sever to communicate with the Database, the database information was received from Config.py, and the curser connected to the Database was designated as a class global variable using the psycopg2 library. In addition, an execute function was written to unify the result of the query statement. In the case of UPDATE/INSERT/DELETE, commit was made, and in the case of SELECT, values were received. Using the variables and functions, the contents necessary for sebot service were written as a function inside the class.

ros_ utils.py As a class for connecting the server to the robot, the roslibpy library was used to receive the robot's IP address so that it could be connected to the rosbridge_server of the robot. The class received the robot's location in real time using the topic, and the service was used to give the robot a destination, or on the contrary, to receive an image from the robot and store it in storage.

cloud_ utils.py A class for communicating server and storage was created, and a function of uploading and importing files was implemented. To this end, the bot3 library was used, and when uploading a file, the datetime library was used to designate the file title as the current time.

Frontends The web application screen for robot control is made based on React js. Considering that various functions can be repeatedly executed, the application was designed in a way that changes the content of the page quickly. SPA (Single Page Application) was created by replacing components inside the react, rather than loading every single http page from the server. In this project, the application invokes other components and navigation bars that function from Start.js and Start_Admin.js as the default framework. Each time you change the function, the page is drawn faster on the screen because you change the components without getting a new page from the server.

Logins Instead of implementing the login function directly at the front end, the login function and the registration function were implemented through the flask backend. We store the registration form in DB , which is sent by the front end, after checking duplicated forms. After logging in, the session contains login-related information, which allows to connect the robot-related DB or to implement several front end functions.

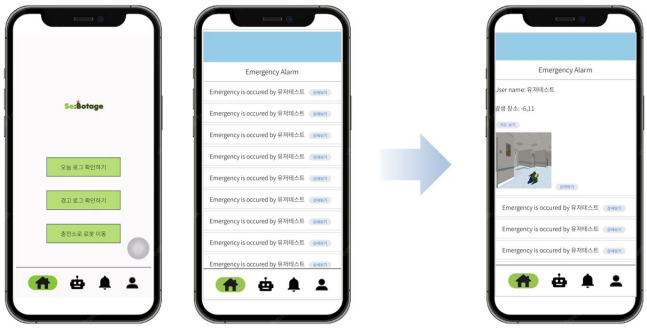


Fig. 4. Application design for Emergency call Service

Emergency call We created the emergency alarm using a browser alarm function called Notification. The robot and the emergency are transmitted through the socket, and when the robot checks the emergency situation, it sends a signal to the front end that the emergency has been occurred through the socket. When the notification function at the front end receives a signal from the socket, it displays an alarm in the form of a pop-up window. It also shows images of emergency situation and its location, which are sent by the robot.

2.8 Challenges

Robotics For robotics we have difficulties in making fall-off detector, we will talk more detail in Implementation section. Since it is really hard to make algorithms that have invariant performances among all environments, we conduct tests for different environments in Gazebo, as well as real-images. In this paper, we finally proposed fall-off detector with assumption of the flat surface. We also discuss its limitations on Limitations section. Also, we made a lot of efforts on identifying our own users from other humans. It is an essential because if a robot identifies a wrong person, it cannot accomplish strolling missions. We use sort track as a solution, recognizing our user with bounding boxes from YOLO and distinguishes them with indexes. Details discussed on Implementation section.

Backends First, the connection between the server and the robot. In general, in the case of robot development, all data is recognized, calculated, and controlled by one computer. However, developing it in the traditional way would have to be open to all port numbers, which could lead to major security problems. To solve this problem, we used websocket communication using roslibpy library, and it was difficult to change the existing code to a way that websocket communication was possible.
The second difficulty was the problem with overhead. Existing robot development is repeatedly received for real-time communication. However, in developing a service robot like this project, we had to solve the problem of overhead if we

wait for user input to come in. After many attempts, it was developed to reduce overhead by using service communication in ROS.

Frontends In frontend part, a web application was planned in such a way that only components were changed without page reloading for comfortable use. However, it was difficult to apply the websocket communication method to this. Planning a service using only components had the problem of opening up too many channels each time a component was reloaded while browsing. Therefore, the page that opens the web socket through page classification using the router is set to be accessible only once. In addition, the number of web socket channels was prevented from unnecessarily increasing by clearing the listener for each component.

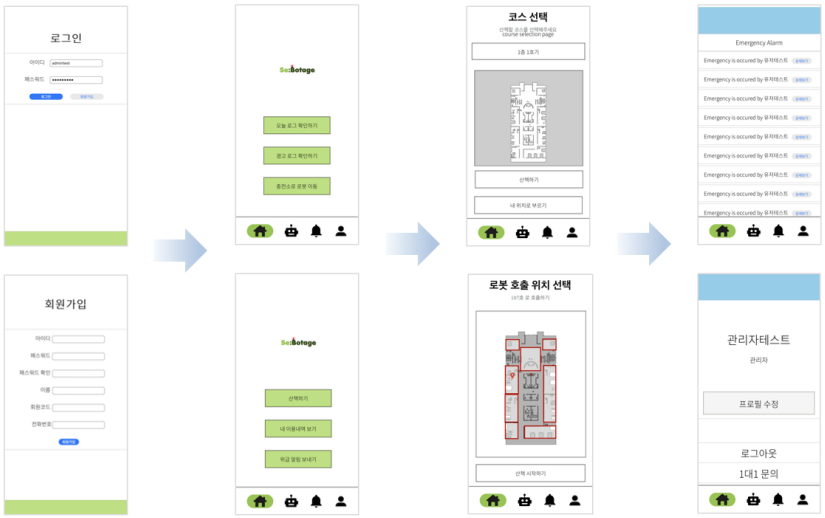


Fig. 5. UI of the application

UI,UX and layout As shown in the Fig. 5, the UI (app screen) was configured. Users can access the app through login membership, and later, UI was created by distinguishing other apps depending on whether they were administrators or not. The focus was on creating a UI screen that is not as complex as possible and contains only the necessary information neatly.

The flow chart in Fig. 6 expresses how the app is implemented in operation as a whole. Through login, the member_info DB is accessed and the session is retrieved, thereby retrieving basic membership information in the web page. In the user app, the part of the robot call is included, and in the admin app, the part of the emergency detection part is included.

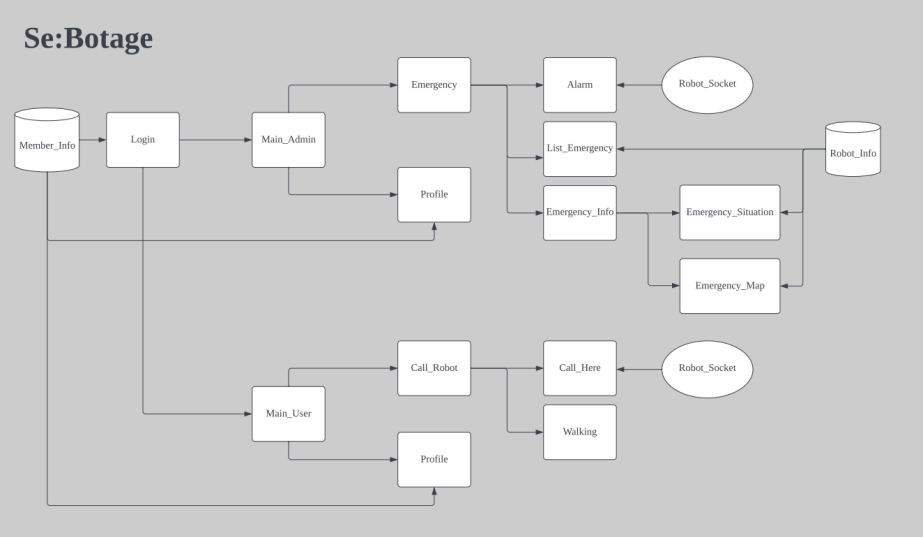


Fig. 6. The main flowchart of the application

3 Implementation

3.1 Robot Call

Robot call is the most basic function of our robot service operation. The call function specifies a robot who receives the command and continuously checks the status if the robot reaches its destination. The second row of the flowchart in Figure 5 shows the robot call screens of the user app. On the screen, when we press the *robot* button on the downside. On the next screen, you can call the robot to the location by specifying the location . The server sends the robot's information and destination from the front, and the server sends if the robot is available. After receiving the signature, the robot's location is continuously updated until it arrived at its destination. We also visualize its location on the user's screen via web socket. When the robot approaches the destination by a certain distance, it sends an arrival signal. The *Start Walk* button is activated on the user screen so that the next step.

3.2 Emergency Detection Service

We have created an Emergency Detection service that detects people falling on the ground and informs managers that he or she is an emergency. This allows the robot to detect a falling motion and to take an action immediately. Figure 7 shows that a person has fallen and the camera has detected energy in the Gazebo simulation environment.

First, we analyzed the image received by the robot's camera and constructed an algorithm to detect the fall-off motion by Openpose; we got the "Keypoints"

Fig. 7. Emergency detected sign (on the top-left corner) for a fall-off motion.

of people like the head and knees. We only use keypoints of head, waist and foot to detect the fall-off motion. Key algorithm consisted of a total of three processes:

Emergency situation

1. A full body of a user should be visible (from head to toe).
2. A head should be under a certain pixel.
3. The line connecting the points of head and waist should be under certain angle with the floor
4. The line connecting the points of waist and foot should be under certain angle with the floor

Our model detects a fall-off motion if it meets the condition 1,2,3 or 1,2,4, specifies in Algorithm1. It returns true when a fall is detected, and false otherwise. It firstly checks if the user's head and foot points are printed successfully, and makes sure that the entire body is captured in the camera. Then, y-coordinates (vertical to the ground) of the head should be below a certain part of the screen, close to the floor. Here, we assume that the robot and a person are on a flat surface, not on a slope. Finally, we analyzed the slope of two lines: the line connecting the head and the waist points (upper body) and connecting the waist and foot (lower body). If one of the slopes of the two lines are within a certain bound and the robot is within a certain angle with the floor, it was judged that it fell. In the experiment (See section Experiment), the head threshold ($head\_threshold$) was set at half the height of the image, and it was set to be detected if the angle was within 60 degrees from the floor (min_angle , min_angle).

When the robot detects an emergency, a signal enters the front end through the socket, and alerts the administrator that the emergency has been detected by a pop-up window. The robot sends pictures and locations to the database in an

Algorithm 1 Algorithm for Fall-off Detector

Require: $head\_threshold$, min_angle , max_angle
  procedure FALL-OFF-DETECTOR($head\_point$, $waist\_point$, $foot\_point$)
    if Either $head\_point$ or $foot\_point$ are $None$ **then**
      return $False$
    end if
    if $head\_point.y \leq head\_threshold$ **then**
      $upper\_body\_slope = \frac{head\_point.y - waist\_point.y}{head\_point.x - waist\_point.x}$
      $upper\_body\_angle = \tan^{-1}(upper\_body\_slope)$
      Compute $lower\_body\_angle$ in the same manner using $waist\_point$ and $foot\_point$
      if $upper\_body\_angle$ or $lower\_body\_angle$ is within the angle bounds **then**
        return $True$ ▷ bounds within min_angle and max_angle
      end if
    end if
    return $False$
  end procedure

emergency situation.When the app accesses the emergency page, it sends a request with the location and picture record of the fall detection on the emergency page. In addition, when the coordinate of the emergency occurance is delivered to the backend, we can also notify it through the application.

3.3 Stroll Assistant

In addition, we implemented a service to take a walk together while looking at the elderly (user) in an outdoor environment. At this time, a robot is not just move to the destination, but analyzes the location of the elderly in real time with sensors to ensure that the elderly are following properly. We developed "Depth YOLO" using the location of the person from YOLO and the depth data from the robot's sensors to tell how far away the person is, and we implemented an algorithm that identifies the movements of different people through sort tracking.

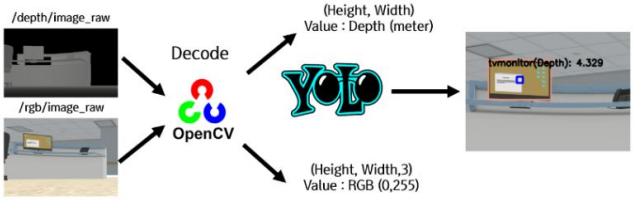


Fig. 8. Depth-YOLO : Aligning Depth with RGB

Figure 8 shows the specific operation of Depth YOLO. We receive RGB and depth images from our robot's sensors. We can obtain bounding boxes of RGB images by YOLO. We used YOLOv3[8] model, which shows good performance compared to the latest ones, and the pretrained weight provided in advance. We also used a light tiny-weight for fast detection. The bounding boxes and depth images are combined together so that we can finally be able to find out the location of the object, how far away from the robot.

We also implemented the Sort track algorithm into the ROS system so that our robot can distinguish between the target users and ordinary people when driving. Sort tracking uses the bounding box obtained from YOLO as an input to analyze the movement of the object in consideration of the position change of the bounding box and index it as shown in Figure 9.

However, we have to determine which index belongs to our user now. In addition, indexes often changes; the user is often recognized as different person (index). This is because the user is often lost from the camera screen for a while and other technical issues. So in this situation, we had to implement an algorithm to determine how to get the user's index back and drive.

We divided the walking service into two states (See Figure 9) A robot knows the user's index in the tracking(idle) state. If the robot cannot find a user with that index for a period of time, it enters the identification state. In this state, the robot moves very slowly, saving the user's index again. The robot stores the index of the closest person within a certain range of people and recognizes it as a user. When it finishes identification, it returns to the tracking state and drives again . If you look at the bottom left of Figure 9, you can see the index of the user recognized by the robot in the bottom left. In the tracking state, you can see that the robot has successfully marked the user's index "7", and has changed to "None" in the identification status in the right.

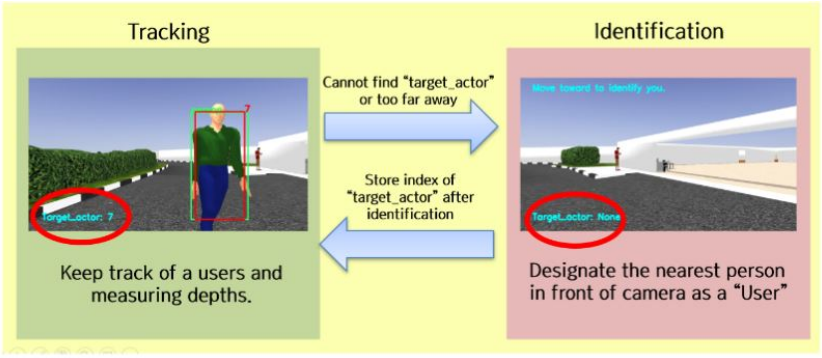
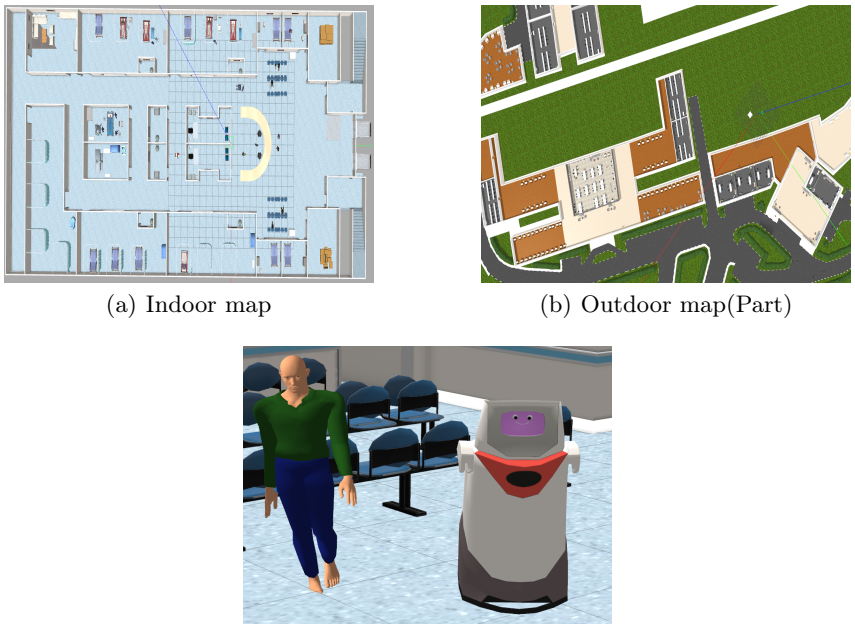


Fig. 9. Two status of Strolling Service

4 Experiment



(a) Indoor map

(b) Outdoor map(Part)

(c) Actor and robot

Fig. 10. Basic configurations of the simulated environment

4.1 Configuration

We conducted an experiment in the Gazebo simulator. The simulation environment was conducted in two different environments, indoor and outdoor (See Figure 4(a) and (b)). We conducted three experiments: robot call functionality via the app, sending an emergency alarm after detecting a fall, and taking a walk with the user. The first and second experiments were conducted on an indoor map, and the walking scenario was conducted on an outdoor map. We made a 2D map of two environments using gazebo plugin, and the robot runs and localizes through this map.

We also placed an animated actor who acts as a "user" in the experiment(see Figure 4(c)). We developed gazebo plugin file that is able to manipulate the actor to move free as we want.

4.2 Test1: Robot Call

In the first test, the goal was to send an arrival point to the robot, the robot moved to that position, and the arrival alarm was sent back to the user after

arrival. In addition, it was also confirmed whether the position of the moving robot was continuously displayed on the user's screen.

As a result of the experiment, on the first page, the location of the robot to be selected was accurately displayed on the screen. On the second page, the destination could be specified, but it was confirmed that the information of the selected destination was uploaded through the server. On the third page, while the robot was moving to a designated destination, the location information was continuously transmitted from the robot and the screen was updated. The period at which the screen is updated is set to 1 second to reduce the load on the server. Finally, when the robot arrives at its destination, it stops moving and updating the screen and sends an arrival signal. It was possible to check even displaying this on the screen. We also confirmed that the robot automatically tracks the optimal path to the destination using navigation stacks to track the path to the destination. If an obstacle was found in this process, the robot could reach its destination by modifying it to an appropriate path.

4.3 Test2: Emergency detection & Alarm service

The most important part of the app was to show it to the app when the robot sent an emergency alarm. First, when the robot receives an emergency detection signal that detects a fall and sends it, the app should be able to show it as an alarm, and then the emergency page should be able to express the picture and location of the robot's sender.

So we tested the alarm function and the emergency page by operating the app and the robot at the same time. When a robot detects a person falling while walking on a map, an emergency is occurred alarm appears in a pop-up window in the application at the same time. Entering the emergency page, information was included in the order of alarms that occurred, and when you click on the view of detailed information, you can see a scene of a person falling over the emergency and a map of where the emergency occurred.

At first, there was a problem that when it was operated once, signals and photos were transmitted multiple times in duplicate, indicating the same multiple information as emergency. Since the callback function, which receives the image retrieved from the robot's camera in real time, continued to run even when the emergency call was made, it was determined that several of the same images were transmitted. Therefore, when transferring images to the server, this callback function was interrupted to prevent multiple images from entering at once, which enabled a clean emergency function.

4.4 Test3: Stroll Assistant with Tracking

Our final experiment was conducted in an outdoor environment (See Figure 4(b)). At this time, we checked whether the robot accurately recognizes the user among several people in the camera. In addition, we also watched if the robot

slows down and changes to identification status when the user disappears from the camera or the index changes due to an error in the sort tracking algorithm.

We were able to know the index number of the user and the current status of it through the screen (See Figure 9. We set the environment with multiple people in a strolling route to evaluate the performance of tracking. We set the threshold time as 3 seconds; if a robot lost index of our target user for more than 3 seconds, it becomes the identification status.

As a result of the experiment, the robot successfully arrived at its destination. Also, until arrival, no one else was recognized as a target other than the user. In some cases, the user's index changed while driving, and even at this time, it was successfully re-recognized to change the target index and drive. In addition, the robot slowed down and changed the status when the user not found. When re-identification was completed, the speed was increased again to successfully resume driving.

5 Limitations

5.1 Fall-off detector in emergency service

First, the algorithm assumes that the robot and the user are in a flat place, so it is likely that they will not work well on a slope. This is because the person's coordinates can be affected by the slope, and it includes an expression that compares the y coordinates of the head with the threshold ($head\_threshold$) within the algorithm (See 1).

Also, we're not going to analyze a series of pictures, but a single image. Thus, our model cannot determine whether it detects a real fall-off motion , or just lying on the floor. In addition, there were many cases where keypoints of a person was incorrectly given. Moreover, we have to tune parameters for every change of the environment.

5.2 Sort tracking in Outdoor Strolling Service

Since our stroll assistant only analyzes the movement of humans, there is an inaccuracy of identifying the wrong person as our user. Although we added an identification status, there's a chance of mis-identification if someone interrupt you in front of the camera. Therefore, in order to implement accurate tracking, we need to find it through a method other than sort tracking. We tried to develop the face recognition before sort tracking. We use a model from dlib[9], an open source c++ library with variety of machine learning algorithms. However, we failed to train and use this model since the model can't recognize the face landmarks of our simulated humans well. We concluded that face recognition is inappropriate for simulation environment, and it would be a better option for real environment than sort tracking. Also, it is an alternative to locate the user through GPS, unlike indoors.

5.3 Simulation

In fact, we chose simulation rather to implement service in real-world to avoid the labor of making robots. On the contrary, there was a problem with the simulation environment. There was a problem that our own implemented actors' movement (by a Gazebo plugin) was strange and was not recognized by simulation's LiDAR. We solve this problem by detecting the actor by depth camera instead of LiDAR. Also the faces of several people on simulation were difficult to distinguish.

5.4 Environment

According to our basic purpose, we tried to take the form of a cloud robot by turning a specific node of the robot from the server. However, due to the price problem of the server, only light programs were made to work on the server. Also, as we did most of the computations on the simulation computer, we had a lot of difficulty working on it due to the overload.

5.5 Saving Information in Application

Currently, our app does not store information within React, but continuously receives it through the backend, which leads to the disadvatages of inefficiency when sending and receiving information through the backend. We didn't correct this inefficiency because it was difficult to correct once the operation was started. If we could have stored some information on the entire app through context or Redux in React, we could solve this problem. We could also be able to use global variables within our front project, which would have been more efficient.

6 Future Work

6.1 Improve performance

This project was not only difficult to build robots directly due to the physical limitations of the resources currently available, but also had a lot of difficulties in implementing robots and deep learning in simulation. If there had been better hardware and enough resources to implement robots, I think it would have been possible to build robots, and it would have been a nursing home robot that showed more realistic and accurate performance through additional deep learning technologies.

6.2 Cloud service

It can evolve into a cloud robot that applies cloud services to robots. If we proceed with the implementation of additional functions based on cloud functions, we will be able to use our robots in more diverse places, not in limited places.

6.3 Business scalability

Due to the characteristics of caring labor, it is possible to apply the scalability of robot personnel such as caring for children in kindergartens or helping patients in hospitals. In addition, from the perspective of the app, when adding and improving additional app functions according to each environment, various environmental application functions can be implemented, and user-friendly functions can be provided through more convenient, necessary, and intuitive app functions and design.

7 Conclusion

In this paper, we proposed the mobile robot platform at nursing home with application service due to the demand of aging society. The application have two types: user and admin. Our proposed model has three main functions. The first implementation is robot-call. Users can call a robot in a nursing home by a user app, and the robot immediately moves to the following location, give arrival sign to server when it reaches its destination. Also, it serves the real-time emergency detection, detecting a fall-off motion and send an emergency sign to an admin application. Our final implementation is stroll assistant, which pioneer the strolling path for users. Here, our robot can notify the user among other people through sort track, with our own two-process stroll system of identification and tracking(idle mode). We did an experiment in indoor and outdoor maps of Gazebo simulator, make 2D maps and actor controller which manipulate our human model by plugins. A robot in the simulator successfully achieve our three scenarios. However, our proposed service has several limitations on fall-off detector and stroll mechanisms. Our model can be further developed with using additional techniques such as face recognition, and GPS as well as implementing our server in cloud.

References

1. J. Lee and H. Jeong, "어르신 돌보러 방문요양 왔는데…"밭을 매라고요?," *Hangyeore*, Mar. 2019.
2. Redmon and Joseph, "You only look once: Unified, real-time object detection.," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
3. Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, 2017.
4. Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.
5. A. B. Wojke, Nicolai and D. Paulus, "Simple online and realtime tracking with a deep association metric," *IEEE international conference on image processing (ICIP)*, 2017.
6. G. Welch and G. Bishop, "An introduction to the kalman filter," 1995.

7. B. Sahbani and W. Adiprawita, "Kalman filter and iterative-hungarian algorithm implementation for low complexity point tracking as part of fast multiple object tracking system," *2016 6th international conference on system engineering and technology (ICSET)*, 2016.
8. Redmon and Joseph, "Yolov3: An incremental improvement.," *arXiv preprint arXiv:1804.02767*, 2018.
9. D. E. King, "Dlib-ml: A machine learning toolkit," *The Journal of Machine Learning Research 10*, pp. 1755 – 1758, 2009.