Robotics for Artificial Intelligence – KIMROB03
ALICE - A Domestic Service Robot

Group n. 07 — Sudhakaran Jain and Avinash Pathapati

Abstract— **The main goal of our work is that when the domestic service robot named Alice receives an order containing objects and their corresponding locations, it should successfully navigate to those locations and identify if those objects in the order are present at those corresponding locations. Then it should be able to pick those identified objects and place them on it's basket. Finally, once it has picked up all the objects, it should return to the drop-off location.**

The main achievement of our work is that Alice was able to navigate successfully to the given locations,recognized objects properly to some extent in those locations and grasps the identified objects and places them on it's basket. As Object recognition system is not perfect, Alice some times makes errors in recognizing objects and tries to grasp wrong objects.

I. INTRODUCTION

Domestic service robots can assist human beings in doing repetitive and dangerous tasks in their daily life tasks. One of those mundane task might be taking a medicine. To accomplish it the person has to look and then take the medicine himself. What if a domestic service robot exists which can successfully identify the requested object and can return it to the location of the person himself! This would save a lot of time and effort for humans. Our work is focused on solving such tasks. To accomplish these tasks we need to perform the below sub tasks and a domestic service robot named Alice was used to implement it.

- Navigation: Alice should be able to successfully navigate to the location after finding a shortest optimal path containing no obstacles on its way. Here we used Dijkstra's algorithm [1] to calculate the optimal path.
- Object Recognition: Once the goal is reached, Alice should be able to successfully identify the target object and also predict the position and orientation of that object. A Convolutional Neural Network(CNN) was trained and used for identifying the object and another pre-trained neural network is used for finding the orientation.
- Grasping: Using the predicted position and orientation, Alice should be able to grasp the object properly so that the object does not fall down from it's fingers and put it on it's basket.

In final demo, Alice needs to collect orders in a warehouse for someone's shopping list. There will be five different types of objects namely Base Tech, Tomato Soup, Eraser Box,USB Hub and Ever green. These objects can be present at any of the two tables present in the Warehouse. An order of items which contains the list of objects to be picked and their corresponding locations is sent to Alice. As soon as Alice receives the order, it should be able to navigate to each table and pick all the objects if they are present in the order list for that particular table and put them in it's basket. Finally, Alice should be able to successfully navigate to the drop-off location where a human worker will collect those items from the basket.

We will start by presenting our methods in detail which we used to solve this task. Later, we will see the results of the experiment when our approach is tested both in simulation and on Alice. Lastly, we will discuss and conclude on the reasons for the observed results followed by possible approaches which can further improve the results obtained.

II. METHOD

The robot Alice has various sensors attached to it which mainly includes:

- Laser: It is used by Alice to create the cost-map of its surroundings. All the obstacles present nearby are recorded in this map, hence the name cost-map.
- Front/back camera: It is used to see the objects present near Alice.
- Wheel encoders: It is used to measure odometric data to estimate the current position of Alice.
- Robotic arm: It is used to grasp an object from a location and put in the basket.

Figure 1 shows how Alice looks.

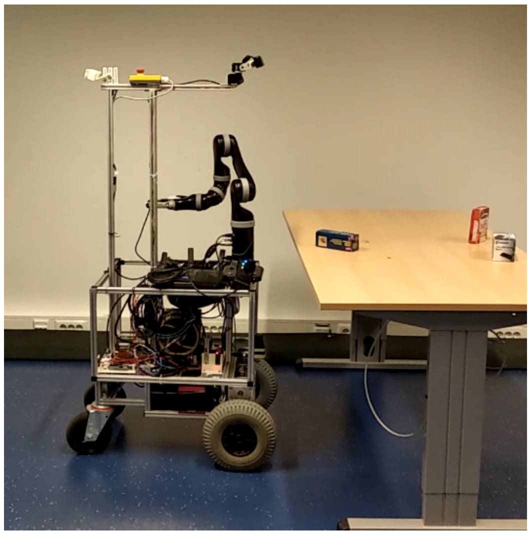

Fig. 1. Domestic Service Robot Alice.

The authors are with Faculty of Science and Engineering, University of Gronignen, The Netherlands. {s.j.jain,a.pathapati}@student.rug.nl

The whole implementation was done using ROS (Robotic Operating System) [7] and programmed in Python. Gazebo was used for 3D simulated environment and Rviz was used to visualize what Alice sees. As mentioned earlier to successfully accomplish the task, Alice has to perform three sub tasks which are explained in detail below.

A. *Navigation*

To navigate successfully to a goal node, Alice follows the path given by a 'global planner'. The global planner contains the main algorithm needed for calculating the shortest optimal path having no obstacles to reach the goal. But, this Global planner requires a global cost-map of the environment to calculate the best route. So a global cost-map was created by making the robot roam in simulated environment while the laser sensor records all the obstacles nearby assigning some cost value to each region in the map. In our case if a particular cell doesn't contain any obstacle,the cost value of that cell is zero. Using this cost-map, the global planner generates a path with minimum cost from source to destination. Our aim was to keep the main algorithm of the global planner as simple as possible but efficient enough to perform well in the experiments. Hence, we chose Dijkstra's shortest path algorithm as it met the above mentioned criteria.

We now briefly explore how Dijkstra's shortest path algorithm works:

1) Mark all the cells in the grid map as unvisited and create a set which contains all these unvisited cells.
2) Assign tentative distance as zero to the source and Infinity to all the other cells in the grid and set the source cell as the current cell.
3) For the current cell we fetch all the neighbouring cells which doesn't contain any obstacles in the grid map and compute the distance to those cells from the current cell. If the computed distance is less than the assigned distance for a given cell, we replace it with the computed distance and make the current cell as parent for that corresponding cell.
4) Mark the current cell as visited by popping it out from unvisited set. If the destination cell is visited or if there is no path from the current cell to destination cell, then the loop stops.
5) If not, select the unvisited node with the minimum distance and set it as the current cell. Then go back to Step 3.
6) After the loop stops, we keep backtracking the parents starting from destination cell till we reach the source which ultimately gives the shortest optimal path.

The navigation also depends on one more object called 'local planner'. While navigation, the robot is liable to see few more obstacles which may have not been present in the global cost-map. Here is where local planner having local cost-map comes into picture. This local cost-map contains the current local obstacles in the near by surroundings of the robot. This helps the robot to have better knowledge about its surroundings and trace a path to the goal without colliding with obstacles.

To further ease out the navigation for Alice we also provided intermediate waypoints. So using these waypoints as the intermediate nodes Alice navigates to the goal. While setting up waypoints we realised that objects can be present on both the end of 'Table2'. So we provided two different waypoints to either side of 'Table2'. Even after this, we still believed that there might be practical hurdles where Alice wouldn't be able to estimate an optimal path for its navigation. To overcome this problem we also added a adjusting mechanism where Alice moves back by 0.4 meters whenever it is not able to find an optimal path to the goal. After moving back, it again tries to find an optimal path for the goal.

Once the final waypoint for the table present in the order is reached, an algorithm called 'alice approach' is invoked. The main function of this algorithm is to check the presence of objects in front of Alice. If any object is found, Alice will navigate in the direction of the object. The algorithm makes sure that Alice doesn't collide with the table on which the object(s) is kept and maintains a minimum distance from it. Finally, Alice aligns itself along the orientation of the table getting ready for object recognition.

B. *Object Recognition*

After Alice approaches a table, it has to recognize all the objects present on that table. The front camera of Alice keeps sending the ROIs (Region Of Interest) as images which can be used to recognize the objects present in front of Alice. Our idea was to use a Convolutional Neural Network(CNN) [2], [3] using tensorflow [5], [6] library to recognize these images. The architecture of a CNN is shown in Figure 2. To train this CNN, we were initially given a dataset containing 200 images along with their dimensions for each object class namely Base Tech, Tomato Soup, Eraser Box,USB Hub and Ever green. But we realized that we won't be able to achieve good accuracy results if we train the CNN only with these 200 images for each class. Hence, we decided to do perform data augmentation to increase the size of dataset.

- Data Augmentation: This involved scaling the images from top, bottom, left and right by a factor of 0.2. More images were augmented by increasing the brightness of these scaled images by a value of 30. The final dataset thus obtained had a around 2000 images for each class. The Figure 3 shows how the augmented images look like.
- Convolutional Neural Network: The input size of the image was 32*32*3. So we decided to use two convolution layers one having 32 filters of size 8*8*3 and other layer having 64 filters of size 5*5*32. We used 'RELU' as the activation function. Two pooling layers were put in between these convolution layers and two fully connected layers were appended in the end.
- Training CNN: The final augmented dataset was used to train the CNN as well as for validation testing. The data was fed in batches and validation testing was done after training each batch to calculate the accuracy of our

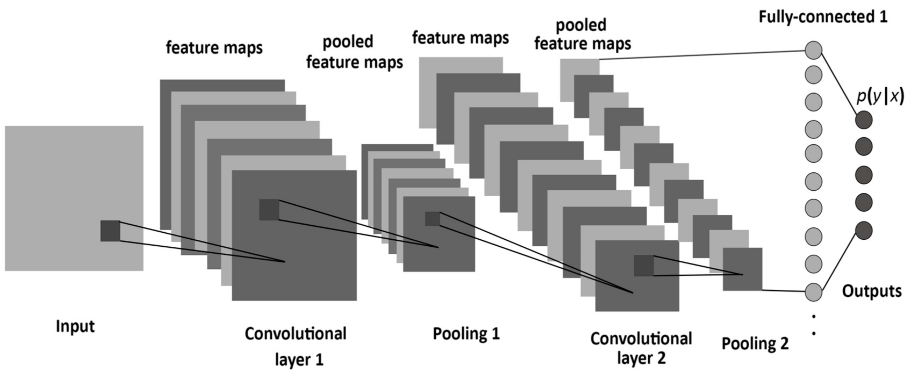

Fig. 2. Architecture of Convolutional Neural Network.

model. We received a validation accuracy of 100% after around 100 epochs.

- Testing CNN: Finally, we tested our CNN with the test data which was provided in a rosbag file. Our CNN model attained an accuracy more than 90% in this test data.

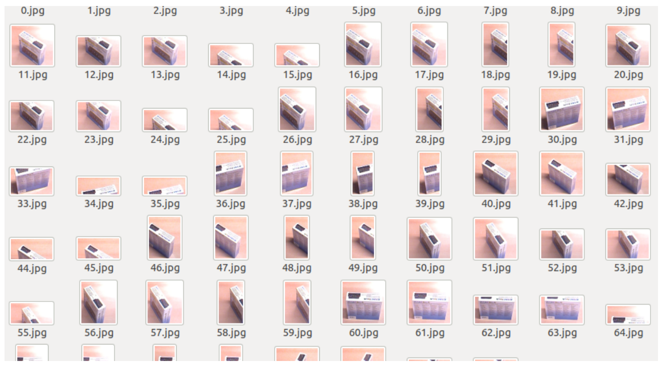

Fig. 3. Augmented dataset

C. Grasping

Once the object recognition for all the objects in a table is done, Alice will try to pickup objects specified in the order from different positions of the location because if the objects are located further than the arm could reach, it couldn't grasp. The grasping of the object is implemented as below

1) Move the arm(end effector) close to object
2) Open Fingers
3) Translate to object
4) Close fingers
5) Attach object to end effector
6) Translate up

Most of these functions are handled by built-in 'pick' [4] method. We had to provide the dimensions and orientation of the object to be picked. As Alice would be grasping only the objects among the five as mentioned, we hard-coded the dimensions for all these object classes. Another pre-trained neural network model which takes an ROI image as input and predicts the orientation was used to obtain the orientation of the object to be grasped. Before Alice could actually grasps the object, it creates a plan to execute it. If Alice finds that there is possibility of collision during planning, the execution is not performed. To make sure that Alice won't collide with the object it is picking, we tried providing different positions which vary in depth so that if at least one of them is collision free, Alice would try to grasp at that particular depth. Once it grasps, Alice moves it's arm to the position of the basket and drops the object in the basket by releasing the fingers. The same process is done for all the objects it recognized from the order for that particular location.

D. Behaviour

Figure 4 shows the state machine diagram for our implementation. Before Alice is ready to receive an order, it moves to the start location. Once reached, Alice state will be 'initial' before it receives an order. Once Alice receives an order, the state of Alice is changed to 'sub1' where it will navigate to 'Table1' first if it is present in the order. Once Navigation is completed successfully, the state of Alice is changed to 'sub2'. If Alice gets stuck during navigation the state of Alice is changed to 'Adjust' where Alice would move 0.4 metres backwards to overcome collisions and path tracing problems. Once 'Adjust' is done, the state of Alice is changed back to 'sub1' during which it tries to navigate again to 'Table1. This process is repeated until Alice successfully navigates to 'Table1' and 'alice approach' algorithm is run. Then the state of Alice would change to 'sub2' where Alice would be performing both object recognition and grasping. So if Alice identifies any of the objects present in the order at 'Table1' it tries to grasp and put it in it's basket. Once this is done successfully, state of Alice changed to 'sub3' and Alice navigates to 'Table2'. Now the same process mentioned above is repeated except that Alice moves to both ends of 'Table2'. After this step Alice finally moves back to start location.The state of Alice goes to its 'initial' and waits for a new order.

Apart from this, we added a text-to-speech facility in Alice, so that while executing various sub tasks, Alice informs us through speech, what next move it is going to perform.

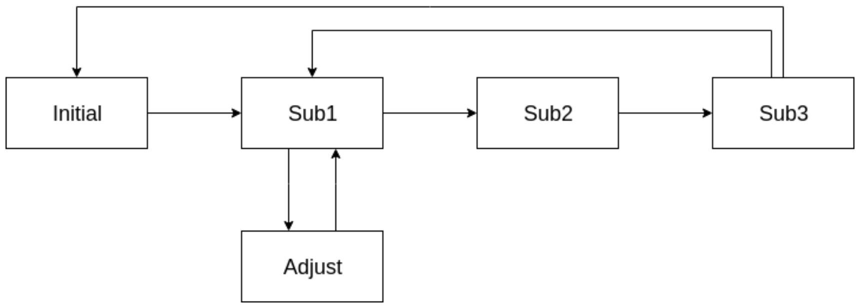

Fig. 4. The Alice state machine.

III. EXPERIMENTS

We tested our implementation of the task in both simulation and on the domestic service robot Alice. The experimental setup consisted of 3 difficult orders to be completed.

1) Alice will need to pick up 1 object from a known location. This object will be present with 1 other object present as well. The item needs to be returned to the drop-off point.
2) Alice will need to pick up 2 or more objects from either table 1, table 2, or both tables. The object(s) may or may not be there. The objects (if found) need to be returned to the drop-off point.
3) Alice needs to clear both tables in the correct order, so either first table 2 and then table 1 or the other way around. The items need to be returned to the drop-off point.

The result of execution of these orders for the three sub tasks namely Navigation, Object Recognition and Grasping in both simulation and real world are explained in detail below.

A. Navigation

In both cases of experimenting in Simulation as well as in real world, first we had to create a cost-map through which Alice will navigate. So in case of simulation we captured this map by moving the robot in a simulated environment using Gazebo. In real world we used Joy Controller to move Alice. As we were inexperienced in using a Joy Controller, for us capturing the map in simulation was relatively easier and faster compared to capturing it in the real world.

As the map used for simulation and real world are entirely different, the waypoints used during navigation were also different for both of these. So these waypoints are captured separately in both the cases using Rviz. Figure 5 shows how waypoints look in both the maps.

Capturing waypoints in simulation was pretty much a cake walk. But, we found it difficult to do the same in real world on Alice. This is because, in real world we had to accurately check which waypoints is the best to navigate to the respective tables. Even if the waypoints are off by a small margin, Alice will struggle to move as it sometimes takes a weird orientation/path than the optimal way expected to reach the goal. In both cases of simulation and real world, we capture the below number of waypoints.

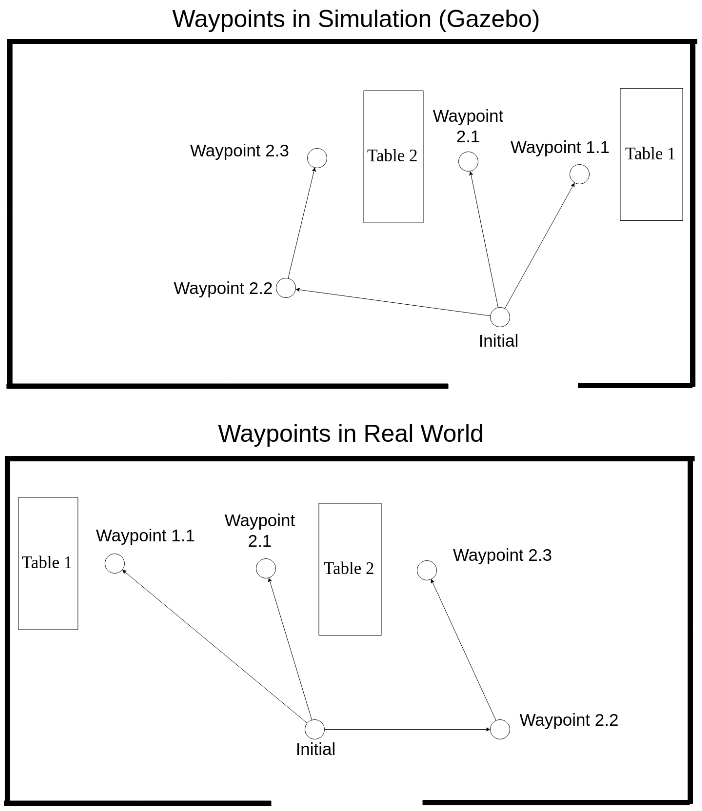

Fig. 5. Waypoints for both the maps.

1) one waypoint for the initial position
2) one waypoint for ’table1’
3) two waypoints for ’table2’

When the order was given in simulation, we observed that the robot moved to the respective table with ease and started doing the next sub task. Whenever, it couldn’t find an optimal path, it went into ’Adjust’ state moving back by 0.4 meters and again resumed to find a path to reach the goal.

When an order was given in real world, we saw that Alice took some more time to trace a path to the goal node. We then realised that our waypoints in the real world map was unfortunately not very accurate. Alice went into ’Adjust’ state more often to overcome path finding problems. This was mainly because of two practical issues which we noticed.

Firstly, we noticed that the obstacles seen in the local cost-map of Alice were more nearer than they were actually present. Due to the fear of colliding, even if the obstacles were a bit far away, Alice kept on finding different paths to the goal till it found a path it felt that won't collide. At times, we also had to manually clear the local cost-map if Alice was unable to find a path for a longer time. Secondly, we discovered a fault in our 'Adjust' mechanism. Whenever Alice could not trace a path to goal, it simply went back 0.4 meters to search for a path again. But here we didn't take into consideration if there was an obstacle located just behind Alice. These factors made Alice to take more time in finding the paths to the tables.

B. *Object Recognition*

When we tested our network with real ROI images from rosbag file, we got an accuracy of more than 90%. But when we tested the same images on fully trained Google's Inception network, we received comparatively less accurate results. We realised that this must be because the Google's Inception has been pre-trained for a higher resolution of images and it expects extensive number of feature maps which our low resolution(32*32*3) images cannot provide.

In simulation, after Alice approached the table, its camera started sending the ROI images of whatever Alice sees. We observed that the ROIs not only contained images of the objects lying on the table, but also some images of the table itself which were misclassified as ROIs by Alice. Because of this, our object recognition algorithm started to run on these images too. As a result, some faulty ROI images were recognized as objects that were present in the order to be grasped. Ultimately, Alice tried to grasp these objects which were actually table edges. Moreover, ROIs from the objects that were lying on the other side of the table were also detected.

When the order was executed in real world, all the issues of simulation made their presence here as well. But this time Alice not only took faulty ROI images for classification, but also couldn't make a plan for grasping the objects.

C. *Grasping*

As mentioned before identifying and estimating the orientation of the object to be picked is done using CNN and pre-trained neural network respectively. The position and orientation of the object necessary for grasping is provided using 'grasp pose' method. For a given object multiple grasps at different possible depths like $[z, z+0.25*(size[2]/2), z+0.5*(size[2]/2), z+0.75*(size[2]/2), z+(size[2]/2)]$ are made to make sure that if one of these moves are collision-free Alice will try that move. Here *size* is an array containing 'length', 'breadth' and 'height' of the object. The value z is the height of Alice's base from the ground.
During simulation grasping didn't work exactly the same way each time we tested for the same scenario. Sometimes Alice was able to identify the correct object and grasps it properly and drop it on it's basket. But in other cases it incorrectly identified table edges as objects and tried to grasp them. Because of this, it's arm sometimes got stuck between the table edges. Few times, it dropped the object in mid way after grasping it.

The same issues were present even while testing in real world. But the only difference is the arm couldn't find any plan to execute grasping for any of the objects.

IV. DISCUSSION / CONCLUSION

We now briefly discuss the reasons for the observed results in the experiment we conducted.

Firstly, the waypoints defined for the real world map was not as accurate as it was required to be. Because of this Alice couldn't navigate to the respective table efficiently. It got stuck between obstacles due to which ultimately finding an optimal path took longer time than it should require.

Secondly, to overcome the problem where Alice is unable to find an optimal path, we introduced 'Adjust' functionality in our work. This 'Adjust' functionality moves Alice 0.4 metres simply backwards and then Alice tries to navigate once more from the new location to the goal. This eased navigation in some cases but the scope of the functionality is limited. The main trouble arose when there were obstacles just behind Alice.

Also, because of some weird reasons, the obstacles in local cost-map of Alice were observed to be closer than they actually were.

We also observed two kinds of unwanted ROI images during object recognition. The ones where table edges were treated as ROI images and the other ROIs of far away objects. Both the issues still persisted even after adding a depth factor in our grasping mechanism.
Looking at the practical issues we faced we can propose possible solutions for them to be implemented in future. They are as explained below.

The 'Adjust' functionality can be further improved by making Alice move in the direction where no obstacles are found and try navigating to the goal from that location. This can be done by the using the same Dijkstra's algorithm to move to a nearby temporary location by referring to local cost-map. From here, we can again compute the optimal path to the final goal.

Due to some time contraint, we were unable to check whether the accuracy of the waypoints in our real world map were up to the mark or not. So, if the provided waypoints are accurate without any obstacles in between, then we can expect Alice navigate successfully to the destination with ease in both simulation and real world.

The unwanted ROIs from objects far away can be removed by referring the y-axis coordinates of those ROIs. If the y-coordinates of the ROIs are greater than a threshold value, then those ROIs will be filtered out. Our depth values which we provided were unable to filter out ROIs of table edges. But with more trials we seek find the accurate depth values to solve this problem.

To conclude, this paper explored our whole implementation to make a domestic robot named Alice perform the daily tasks given to it. We explored the differences in the results when the implementation was ran in simulation as well as in real world. We also discussed the practical problems we discovered during experiments and how we seek to rectify them in future.

V. Youtube Video Link

https://www.youtube.com/watch?v=rcQod3O4g18feature=youtu.be

References

[1] Dijkstras shortest path algorithm https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[2] An intuitive guide to Convolutional Neural Networks https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[3] Convolutional Neural Network Architecture https://www.mdpi.com/entropy/entropy-19-00242/article_deploy/html/images/entropy-19-00242-g001.png

[4] Pick and Place Tutorial http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/pick_place/pick_place_tutorial.html

[5] Get Started with TensorFlow https://www.tensorflow.org/tutorials/

[6] Build a Convolutional Neural Network using Estimators https://www.tensorflow.org/tutorials/estimators/cnn

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