EECS 467 Final Project Report
ROBERTO: Autonomous Trash Collection

Aakash Patel
University of Michigan
aakashdp@umich.edu

Bohao Zhang
University of Michigan
jimzhang@umich.edu

Pengqi Lu
University of Michigan
lupengqi@umich.edu

Zhizheng Liu
University of Michigan
zzliu@umich.edu

Abstract

In this project, we present a simulated robot system that is capable of autonomously cleaning trash and clothes from a bedroom environment. We use Mask R-CNN to build a classifier to detect and recognize various objects in the room and use a manipulator to pick up the objects and bring them to the appropriate drop off points. This simulated system serves as a proof-of-concept for a physical home robot that will assist people who are too busy or physically unable to clean their rooms.

1. Introduction

Nobody likes the boring cleaning work, but it must be done. Then why not enlist a robot to clean your room automatically? There are several types cleaning robots on the market, Roomba being the most popular of them, but these robots generally only support vacuuming smooth floors and cannot clear away large objects such as clothes and trash.

Here we introduce a more intelligent cleaning robot, ROBERTO. ROBERTO stands for "ROBot clEans Room: Trash and clOthes". True to its name, ROBERTO can clean a messy room by collecting the trash and clothes. It can also navigate to the rubbish and laundry bins to drop these items off automatically. ROBERTO can definitely make people's life much easier, especially for us engineers who are too busy to clean our rooms!

Our original plan was to setup a mini bedroom using wood boards like in setpoint challenge and place several paper boxes randomly as tables and chairs in the room. Due to the Covid-19 condition, we have to move all the physical setup to a virtual environment and do simulations instead. Despite the limitations, we successfully finished the project and our virtual robot is able to finish all the tasks in a virtual setting.

2. Project Description

Our robot is able to automatically navigate through the environment in a ordinary bedroom setting. In general, it is expected to find large objects on the floor (here we assume objects are either trash paper ball or clothes), identity the type of the object and deliver the object to its corresponding bins. The robot will keep doing this until all large objects are cleared so that a traditional vacuum robot can take over the remaining job. In this project, we assume that the object settings always remain static. In other words, the objects will not be manually moved or changed and no new objects will be added during our robot's execution.

Algorithm 1 describes the logic of our robot. We first assume that the map of the bedroom has already been saved into the robot. The robot first generates a certain number of 'search points' based on the occupancy of the map. As the image process can be computationally hard for the embedded system of a room cleaning robot, the robot will not detect objects in real-time, but stop at these specific locations, i.e, 'search points' defined by us and take pictures of the surrounding environment for detection. As a result, these 'search points' must make sure that all of the map is covered given the range of the robot's camera. And the 'search points' are sorted in an order that the total distance to traverse all the points is minimized. Most of the bedrooms, for example, the bedroom we use for the demo in this report, are not very complicated. As a consequence, the robot does not need too many 'search points' to cover the whole map. For the demo in this report, the search points are given manually. And we assume that the number of trashes is limited and the trashes are distributed sparsely enough so that the robot is able to successfully move from one search point to another without being blocked.

After the robot arrives one of the search point, it rotates towards different directions to take pictures to detect any potential objects. The number and the distribution of these directions must make sure all of the 2π is covered. Based on the specification of the camera we choose, we decide to use 8 evenly distributed directions $(0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4})$ for the robot to rotate. These 8 pictures from the corresponding are then inputted to a pre-trained Mask R-CNN [1]. The network is able to detect and classify clothes or paper balls.

For any of these detected objects, the robot is able to turn towards it based on its position appeared on the pictures taken by the camera. Then the distance between the robot and the object is measured by an ultrasonic sensor so that the exact position of the object with respect to the robot can be computed. The robot then moves to the object close enough and performs pick up action using its arm. In the demo of this report, as all objects including the robot are simulated virtually, there is no necessity to actually implement an ultrasonic sensor. All of the distances are given directly using the data from the simulator.

Algorithm 1 Pipeline Description

```
Variable Initialization
    DetectedObjects ← {}
    SearchPoints ← GenerateSearchPoints(Map)
Localization
for P_i in SearchPoints do
    Move to P_i
    for θ_i in {0, π/4, π/2, 3π/4, π, 5π/4, 3π/2, 7π/4} do
        Rotate the robot towards θ_i
        Image ← RobotCamera()
        DetectedObjects← [DetectedObjects,MaskRCNN(Image)]
    end for
    for [ObjectType,Direction] in DetectedObjects do
        Rotate the robot towards Direction
        Distance ← UltrasonicSensor()
        Move to the object
        Pick up the object
        if ObjectType = Clothes then
            Move to clothes bin
        else
            Move to trash bin
        end if
        Drop the object into the corresponding bin
        Move back to P_i
    end for
    DetectedObjects ← {}
end for
```

Figure 1. Real-world Bedroom

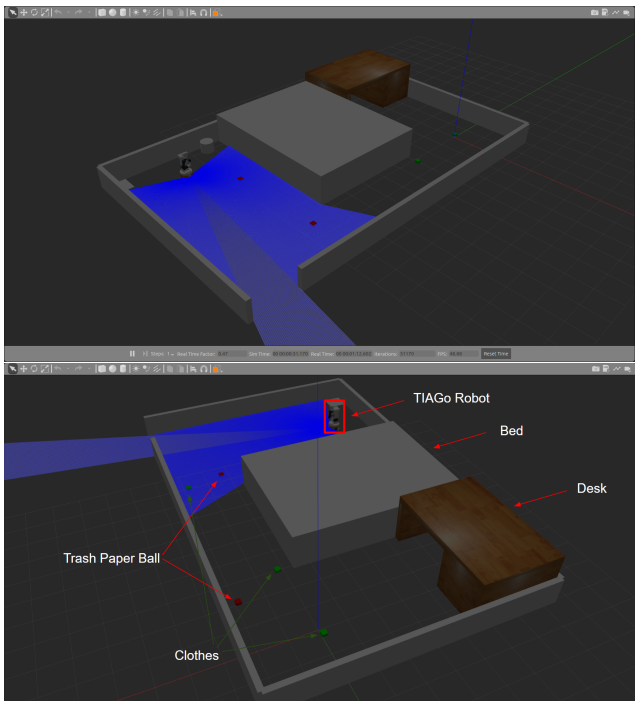


Figure 2. Virtual Bedroom with Robot

3. Project Implementation

3.1. Environment Setup

We use ROS Melodic environment together with Gazebo 9 robot simulator. To maximize the connection between the virtual environment and the real world, we build a virtual bedroom model in Gazebo as shown in Fig.2 based on one of our group member's bedroom as shown in Fig.1. The settings of all the objects, including all the trashes, one bed, one desk and one trashbin are ensured to be the same as those in the real world. Paper balls will be delivered to the cylinder trash bin and clothes will be delivered to a square area on the corner of the room.

Due to the limitations of gazebo simulator, we can only use colored boxes to simulate clothes or trash paper balls, which is unsatisfactory because they are very different from real-world objects. In order to perform object classifications in a more real sense, we separate the classifications task from the robot navigation and arm control tasks. While

Figure 3. TIAGo Robot Model

both navigation and arm control are performed in ROS Gazebo, object classification is performed separately using real-word images taken by ourselves. When taking real-word images, we assume that the bedroom has sufficient natural sunlight from the window and no manual illumination is involved. We also assume that the bedroom is filled with a carpet with one single color, which is significantly different from the clothes or the paper balls we use for the demo. We assume the paper balls are made from folded white letter size paper and clothes are only chosen from T-shirts with single colors.

3.2. Navigation

A model of the Tiago Robot we used in simulation is shown in Fig.3. It has a lidar installed in the bottom base. The hardware of the lidar only supports a 120 degree scan centered at the back of the robot. In addition to the laser scan, the Tiago robot also has an RGBD camera installed on the top of the robot to obtain a 3D information of the environment, detecting obstacles that are higher or lower than the laser scanner plan. The lidar and the RGBD camera can all be simulated in the Tiago navigation package, publishing real-time data on their topics with ideal accuracy no delay within a single scan. As for the odometry and the action model, the simulated robot also has an ideal odometry with little error. However, due to the communication traffic bandwidth, we noticed some delay while transmitting those data from different ROS topics.

We assume the room environment is static with fixed positions of walls and furniture. Therefore we decide to separate the cost map into two parts: global and local cost map. We assume the global cost map is static and known by some laser scan in advance. The global cost map is used for the localization of the robot. Therefore we don't need to update it when the robot moves. However, since there might still be dynamic obstacles to avoid, we will also maintain a local cost map for detecting and avoiding these objects while planning the path.

3.2.1 Localization

Similar to the localization done in project 2, we used the Monte-Carlo localization and particle filters to estimate the pose of the robot by the laser data. However, we adopted the KLD-sampling method as proposed in [4], it measures the approximation error by the Kullback-Leibler distance and bound the approximation error introduced by the sample-based representation of the particle. This method has significant improvement in performance for localization of mobile robots compared to tradition particle filters. The method is implemented in the AMCL package in ROS Melodic and is adopted by the Tiago navigation package.

When the localization begins, we will first initialize the particles filters randomly in the free space of the global map as shown in Fig.6. Then we will publish the global cost map and laser scans to specific topics, and the AMCL package would publish the estimated pose and particle filters back. Since the laser scan is not in all direction, we have to rotate the robot around itself to get a full scan of the initial position to make the sensor and action model work. As there are enough features in the room environment, the particles filters would converge around one pose while the robot keeps rotating. We would stop rotating the robot as long as the estimated robot pose is stabilized as shown in Fig.7 where all the particles converge to the true pose. Then we can start the motion planning to accurately move the robot around the room. As mentioned above, the global cost map is static and we don't need to update it while the robot is moving.

3.2.2 Motion Planning

Our motion planning uses a hybrid approach, which includes a global planner and a local planner using the global cost map and the local cost map respectively. When we want to navigate the robot to a specific goal, we will first call the global planner for a global path. While the robot is moving along this path, the local planner is called based on the local map to avoid dynamic obstacles that may not present in the global map, while trying to follow the global path computed by the global planner.

The Tiago navigation package has provided service for global and local path planning. For the global planner, we have to provide it with the global cost map in ".bmp" format which matches the description file used for gazebo simulation as shown in Fig.4. The package can then generate the global costmap as shown in Fig.5. For the local planner, it will read recent data from the RGBD camera and laser scan and it will build the local cost map around a certain radius of the robot. The Tiago navigation package implemented a similar method for path planning as discussed in [3]. The global path planner uses A* method to generate a series of subgoal points to the target point, and local path planner adopts an improved potential field method to smooth the

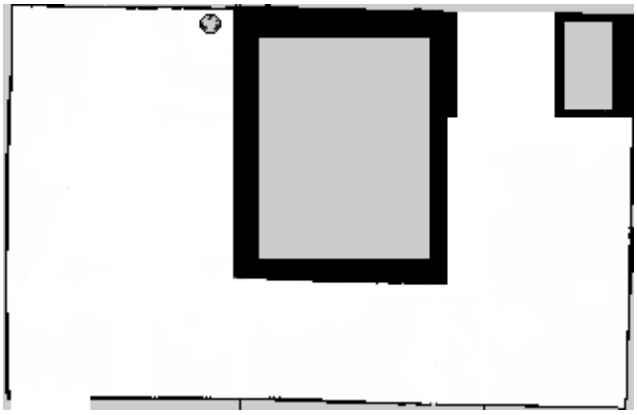


Figure 4. The original bmp file used to construct the global cost map.

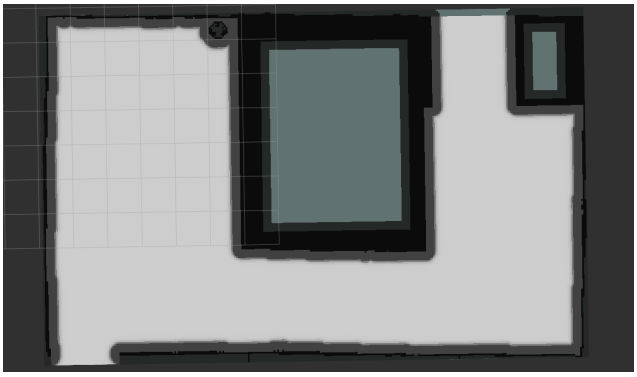


Figure 5. Global Costmap. The white space is obstacle-free and the grey shadow is the static obstacles in the global map. The black lines are the same as shown in the bmp file.

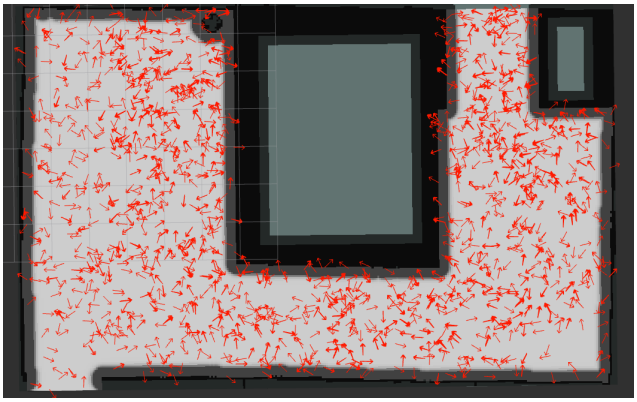


Figure 6. Initialization of the particle filters. The particle filters are initialized randomly in the free space of the global map and each red arrow represents a particle filter with its position and orientation.

path between the pre-planned sub-goal points. This method can not only effectively generate a global optimal path using the known information, but also handle the unknown obstacles in the dynamic environments in time. An intermediate

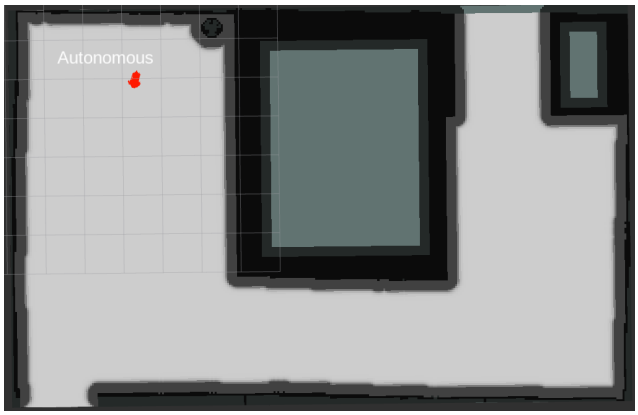


Figure 7. The stage when the localization is complete. At that state the particle filters all converge at the true pose and we can then start the path planner.

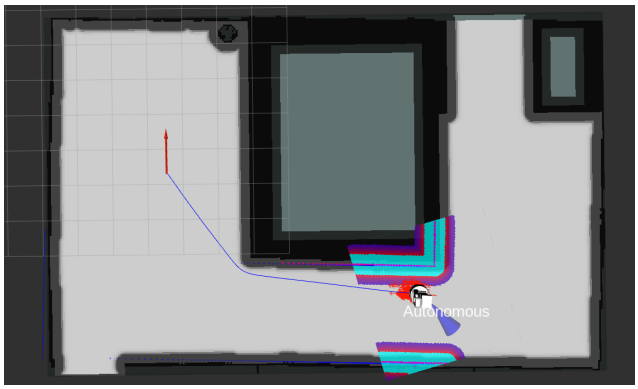


Figure 8. An intermediate state in the path planning.The blue and the purple regions represent the local map and the blue line is the path of the global planner. The blue and red dots along the obstacle are the depth cloud obtained from the RGBD camera.

state of the planner is shown in Fig.8.

Since we have detected the location of clothes and trash in advance. We just need to publish a series of goal points including position and orientation to the hybird planner, and after initialization the navigation will be done automatically.

3.3. Detection & Classification

Our goal is to build a classifier that is able to detect paper balls and clothes in a bedroom setting. To the best of our knowledge, few research has been conducted in a similar environment or towards similar types of objects. The illumination inside a room is highly different from that outside, where most of the objects are evenly illuminated by sufficient natural light. Most of the bedrooms in our daily life have only one window for the sunlight to go through, which makes the illumination setting more sensitive to the position the robot is in. Both paper balls and clothes are highly deformable objects with randomized structures. It could be challenging to train a classifier which is based on

Figure 9. Dataset Examples

a classic CNN structure.

We use Mask R-CNN [5] as the architecture because it is the state-of-the-art model for object segmentation and classification. We didn't find any available dataset that has pictures of clothes and trash paper balls in a room setting, so we decide to generate our own dataset that is more compatible with the need of our project. We use Matter Port version of Mask R-CNN implementation [1] in Tensorflow 1.3.0 and develop our own script to import our dataset and perform custom training.

3.3.1 Dataset Generation

To generate our own dataset, we first drop some clothes and trash paper balls randomly on the bedroom floor to make the bedroom messy. Then we start to take videos from a robot's perspective close to the floor using the mobile phone's camera (iPhone 10, 1080p). When taking videos, we would slowly move and rotate the phone to simulate how the robot would behave. We extract 6066 valid images from these videos. All these images are then cropped in the center and resized to 256×256 pixels. Some of examples from the dataset are given in Fig.9.

3.3.2 Ground Truth Generation

We generate the ground truth using MATLAB Autonomous Driving Toolbox [7]. It is able to track an object labelled by a bounding rectangle with a certain width and height. We manually label the paper balls or clothes appearing in an initial frame with bounding rectangles and use the track function in the toolbox to generate corresponding bounding rectangles in the following frames.

3.3.3 Training

To save training time, we only train the head layers (RPN, classifier and mask heads) and preserve the backbone layer (Resnet101) with weights trained on Microsoft COCO dataset [6] provided by the original paper of Mask R-CNN [5]. Out of 6066 images in total, we use 5000 images for training, 500 images for validation and 566 images for testing. The training takes about 1.12 hours on one GTX 1080 Ti GPU. The specific configurations are listed in Table 1. The training results will be discussed in section 4.2.1.

3.3.4 Further Improvement

Due to the limit on the collection of the dataset and computation resource, we failed to get a perfect classifier as we expected. More details about the performance using only Mask R-CNN are covered in section 4.2.1. We then notice that due to the particular goal in this project, there are several assumptions that we could make use of to directly improve the performance without any neural network techniques. As a result, we add 3 more criterions to eliminate incorrect judgements from Mask R-CNN. Note that the outputs from the Mask R-CNN consist of a collection of bounding boxes which label the detected objects and their corresponding scores which represents the confidence of the results from 0 to 100.

1. If the score of a detected object is lower than 95, it will be abolished.

   The classifier only considers results that it is confident enough with. Otherwise, the results are treated as noise in the background.

2. If the size of a detected object (the area of the bounding box) is larger than 40% of the original image size (256×256), it will be abolished.

   This is an empirical value dependent on the focal length and the range of the camera. Typically a paper ball or a clothes will not appear as a very large shape even if it is very close to the robot. Too large objects will be considered to be impossible for the robot to pick up and collect.

3. If the height of a detected object (the height of the center of the bounding box) is higher than 60% of the original image height, it will be abolished.

   This is an empirical value dependent on position and orientation of the camera. We assume that the camera is located at the bottom of the robot and points horizontally to the front. As the trashes are placed on the

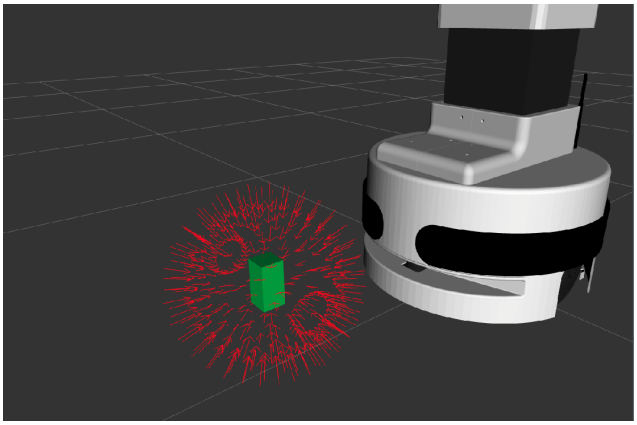


Figure 10. Visualization of grasps for pickup action in RViz

ground so that they always appears in a certain area of the image.

Name of parameters	Settings
Number of epoches	20
Step per epoch	100
Batch size	32
Learning momentum	0.9
Learning rate	0.005
RPN anchor ratios	[0.5, 1, 2]
RPN anchor scales	[8, 16, 32, 64, 128]

Table 1. Training Hyperparameters

3.4. Arm Control

Once the type and position of the object in the environment has been detected, the next step is to use the arm to collect it. Due to time constraints and the difficulty of simulating deformable objects in Gazebo, we simulated both clothes and trash as identical $5 \times 5 \times 10$ cm blocks for simplicity. We use the MoveIt! interface [2] to perform the arm navigation and planning. MoveIt! allows us to access many powerful tools such as motion planners and inverse kinematics solvers. In particular, we chose to use the OMPL (Open Motion Planning Library) planner, which is a probabilistic planning library. Specifically, it uses a rapidly-exploring random tree (RRT) to do the inverse kinematics required for the path planning [8].

Once the robot has detected and chosen an object to pick up, it moves toward it until it is within some fixed distance. The coordinates of the object are then transformed from the world frame to the robot frame. The pickup client then generates a selection of possible grasps to provide the planning interface, which selects one grasp and executes it. Each grasp consists of a pre-grasp approach, which is the direction the arm approaches the object from, the actual grasp pose itself, and a post-grasp retreat, which is the direction the arm moves away after successfully picking up the object. A visualization of these grasps appears in Fig.10. The Once the grasp has been executed, the robot evaluates whether the grasp was successful. We further discuss the evaluation in Section 4.3.

Once the robot has picked up the object, it then moves to the appropriate bin to drop it off in. For simplicity, we model the bins as solid cylinders, and attempt to drop the object on top of them instead of dropping them within a hollow receptacle. To do the placement, the robot performs a similar process to the pickup. First, the coordinates of the bin in the world frame is converted into the robot's frame. However, since we want the robot to place the object on the top of the bin, we use the height of the bin, which is assumed to be known, as the z-coordinate. The planner then generates a selection of possible grasps to reach the target pose, except instead of closing the gripper to pick up the object, it instead opens it to release the object.

4. Results & Discussion

A video demo of this project is available online[1].

4.1. Navigation

In the navigation, we assume the position of the objects to be picked up have given by the detection already, therefore the metric for success of navigation should be how accurate the robot can follow the goal poses. The ground truth of the pose of the robot is published by gazebo and we will compare it to the robot pose obtained from the particle filters. Since a pose has both position and orientation, we will compare them separately. For the position, we will use the euclidean distance in $\mathbb{R}^2$ in the x-y plane. The orientation is represented in quaternions, and we will compute the distance using dot product, suppose we have two quaternions a, b, then the distance is $2arccos(|a \cdot b|)$, which would fall in the range from 0 to π. This distance also represents the difference in radians in the orientation angle.

As shown in figure11, we have recorded the navigation error in all intermediate steps. We can see that in most of the times, the position error is within 0.2 and the navigation error is within 0.1. However, we can still observe cases where the error increase sharply. The maximum position error is more than 0.5 and the maximum orientation error is more than 0.3 (more than 30 degrees). While doing the simulation, the robot would sometimes shift suddenly by a slight amount and it may account for why the error would rise sharply. This could due to that particles filters are less convergent and the resulting estimated pose is not be very robust while the robot is moving at full speed. The communication delay while the robot is moving could also be a factor, as all data are published at a certain frequency and it's hard to synchronize them correctly.

[1] https://www.youtube.com/watch?v=VMEyFwN18oU

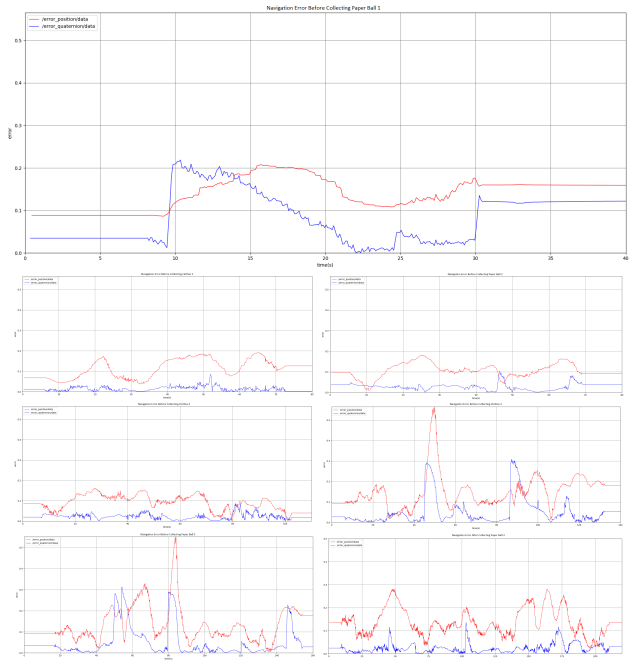


Figure 11. Navigation Errors. The position error is measured in meters and plotted in red and the orientation error is measured in radians and plotted in blue. Each figure represents an intermediate step of the whole pickup and place process.

	Position (m)	Orientation (rad)
Initial Error	0.087	0.034
Before Collecting Paper Ball 1	0.163	0.124
Before Collecting Clothes 1	0.128	0.003
Before Collecting Paper Ball 2	0.092	0.039
Before Collecting Clothes 2	0.041	0.017
Before Collecting Paper Ball 3	0.180	0.053
Before Collecting Clothes 3	0.176	0.031
Final Error	0.138	0.033
Average Error	0.126	0.042

Table 2. Navigation Error at Important Goal Points. The first column shows the error in position. The second column shows the error in orientation.

The accuracy is especially important at the goal points, as if the robot is too far away from the objects while it starts the pickup procedure, the object would go beyond the reachability space of the robot and the pickup would fail. So we have also recorded the errors at important goal points as shown in table 2. As shown in the table, the error is smaller compared to the error while the robot is moving along the path. In addition, the error is very stable that it doesn't become larger as the robot continues to pick up more objects. Therefore we can even add more clothes and trashes in the room to be picked up without failing the accuracy in the end. The average position error is only 12.6cm and the average orientation error is 0.042 radian(2.4 degrees). This error should be small enough for the robot to pick up the object within its reachability space.

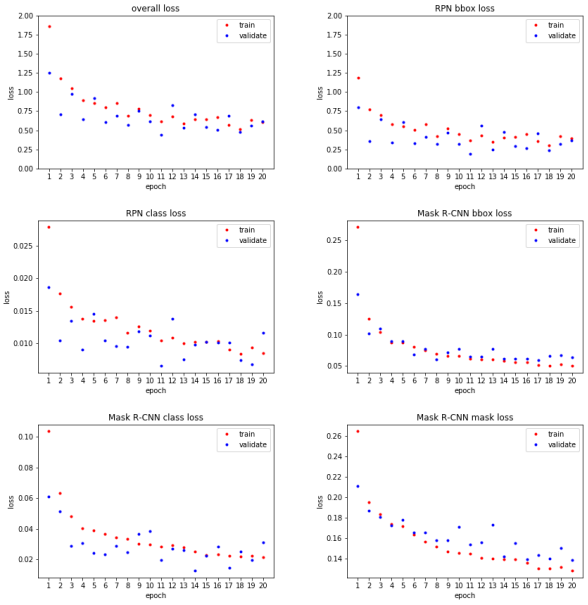


Figure 12. Loss changes

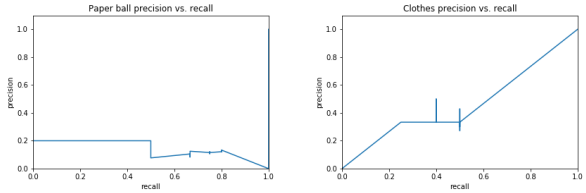


Figure 13. Mask R-CNN precision vs. recall

4.2. Detection & Classification

4.2.1 Mask R-CNN

The losses of the Mask R-CNN with respect to the epoch are shown in Fig.12. The overall loss and losses of different components of the Mask R-CNN are included. We can see that the loss converges successfully through training.

In the demo of this report, we generate 5 search points in total. As a result, there are 5×8=40 images that need to be processed. We evaluate the performance of our classifier on these 40 images. Table 3 shows 6 of the results where our classifier 'detects' something. The detected results are labelled in bounding box together with the type and the score.

The average precision for paper ball is 0.31567 and the average precision for clothes is 0.45238. The mean average precision of our classifier is 0.384025. The plots of recall and precision rate for both types of objects are given in Fig.13.

Observing from the results, we can conclude that the raw Mask R-CNN classifier does not perform well enough for the need of our project. In general, if a desired object (clothes or paper ball) is completely covered by the camera range and is close enough to the camera, the classifier

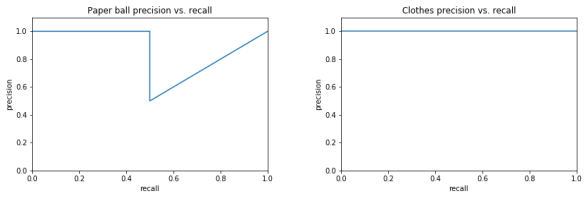


Figure 14. Improved Mask R-CNN precision vs. recall

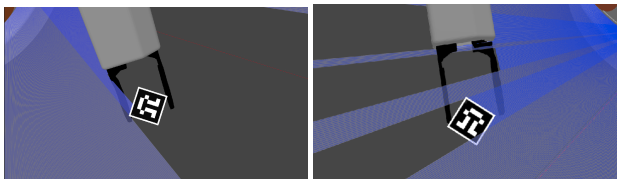


Figure 15. Failed pickup attempts. Left: poor approach angle causes low grip between gripper and object. Right: inaccuracy in moving the arm causes the end effector to hit the object.

is able to detect and classify it correctly. If the object is too far away from the camera and is not detected by the classifier, we have sufficient confidence that it will be detected at the next search point. In other words, given proper search points, the classifier will not miss any desired objects.

The biggest problem of this classifier is that it makes so many incorrect judgements on unimportant objects. For example, as the color is similar, the classifier may believe that some part of the wall or the sockets on the wall are paper balls and the bed sheet is clothes. Note that the bed sheet has never been included in our dataset. As a result, the dataset we generated is still not large enough to take all potential circumstances into account. There are so many unknown objects that could result in incorrect judgement from the classifier. We then introduce some specific measures, which have been introduced in details in Sec 3.3.4, to improve our classifier to meet the requirements of this project.

4.2.2 Further Improvement

We list 6 of results from our improved Mask R-CNN classifier in Table 4 with the same inputs from Table 3. We can observe that the accuracy is significantly improved since most of the incorrect judgements towards the wall or the bed sheet have been eliminated.

For the improved classifier, the average precision for paper ball is 0.84333 and the average precision for clothes is 1.0. The plots of recall and precision rate for both types of objects are given in Fig.14.

4.3. Arm Control

4.3.1 Pickup

The pickup was evaluated based on how often the arm is successfully able to pick up an object at a known pose. A

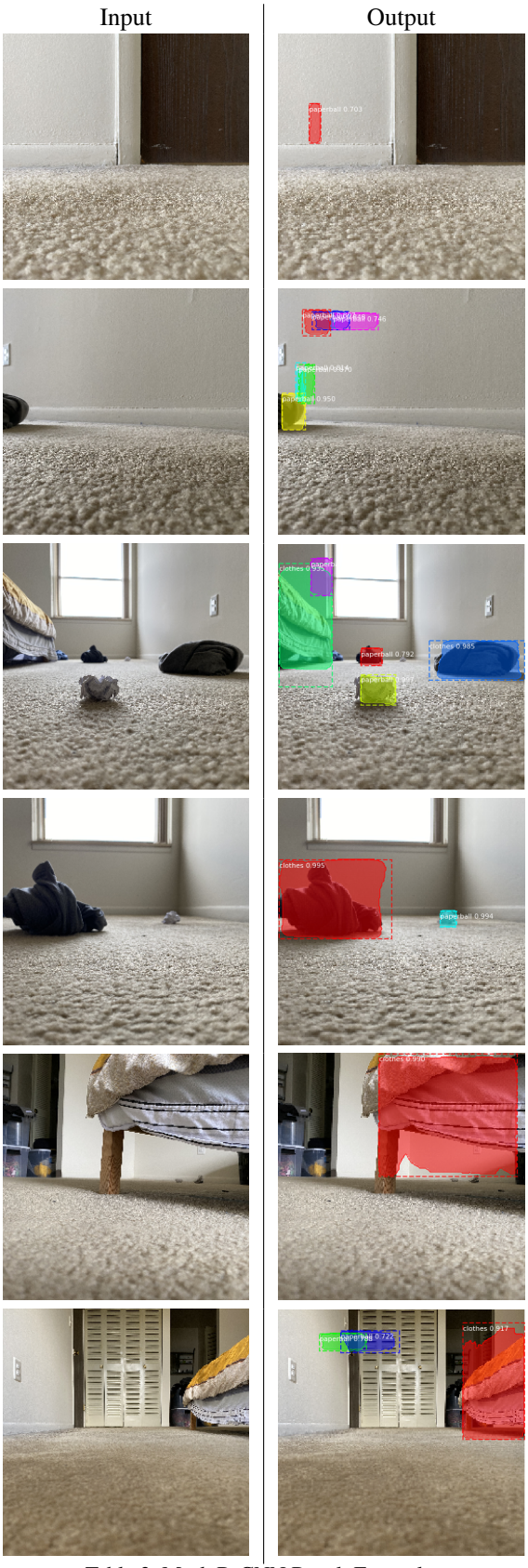


Table 3. Mask R-CNN Result Examples

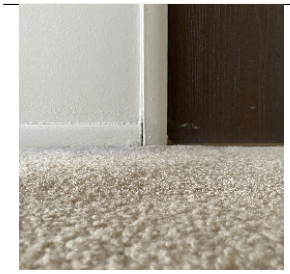
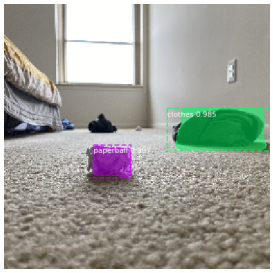
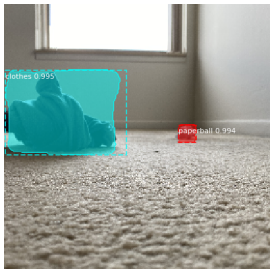
Input	Output
	
	
	
	
	
	

Table 4. Improved Mask R-CNN Result Examples

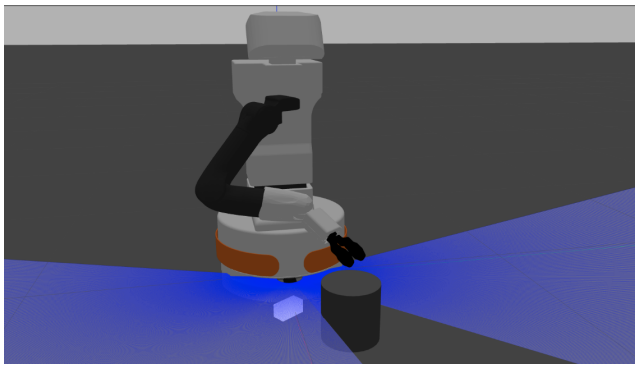


Figure 16. Failed place attempt

successful pickup was defined as one in which the gripper closes around the object, lifts it into the air, and holds it there for at least 10 seconds. In 15 trials, the arm was successful in 11, resulting in 73.3% accuracy.

There are a few explanations for this less-than-perfect accuracy. First is that the objects that we are attempting to pick up are cuboids, with 90° angles and flat faces. If the end effector does not approach the object perpendicular to one of the faces, the gripper will not have much contact with the object, and therefore is more likely to drop it. Second is that although in these trials the pose of the object was known, so the inverse kinematics was guaranteed to be accurate, the movement of the arm still has some variability due to the physics of actually moving the arm according to the PID controller. These two cases are visualized in Fig.15.

4.3.2 Placing

In theory, placing the object ought to be much easier than picking it up, as all it involves is moving the end effector to a particular pose and opening the gripper to let the object fall out. However, we found that placing turned out to be much harder, because the object kept falling out of the gripper before the arm reached the target position. An example of this appears in Fig.16. Solutions to this problem are discussed in Section 4.3.3.

4.3.3 Limitations

There are three main limitations of our current arm control implementation. First, we are restricted to only one attempt to pick up and place an object. If the pickup or place is unsuccessful, such as that shown in Fig.16, our robot is unable to detect the failure and retry the attempt. Realistically, a fallen object can just be considered a new object that the robot has not yet detected. However, in our simulation we make the assumption that no new objects will be added after the initial detection, and so the robot does not return to previous search points. A more robust way of handling this could be to use the robot's camera to track the object from

the time it is picked up to the time it is successfully dropped off, but we did not have time to implement this idea.

Another limitation is that we do not check for collisions in our arm motion planner. In an open environment this is not a problem, but realistically in an actual bedroom setting there are likely to be many more obstacles and items than in our simulation. The robot would need to ensure that the planned path for the arm does not come into collision with any such obstacles, otherwise the pickup or place would be unsuccessful.

Finally, as mentioned in Section 4.3.2, our place planning often drops the object to be placed before the end effector has reached the target location. This can be solved by adding constraints to our path planning that specify certain requirements that the end effector must meet. For example, ensuring that the roll angle of the end effector is close to 0 would help prevent the object from slipping out of the gripper as the arm moves to the target destination.

5. Conclusion

This project presents a simulated robot system that is capable of autonomously cleaning a bedroom environment. Using deep learning networks with Mask R-CNN, the robot is able to detect and classify clothes and paper balls that are lying on the ground. The experimental results have shown that the classifier can quickly and accurately recognize these objects. With use of the robotic arm, the robot can pick up the detected objects and bring them to the clothes or trash bin to drop them off.

Future work on this project would be to integrate the classifier and the simulated robot into a physical robot system, such as the MBot. Although the navigation and arm control code would need to be changed, the principles used in the simulation would still apply. Furthermore, it would be prudent to add support for different types of trash other than paper balls, such as candy wrappers or plastic bottles.

References

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017.

[2] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics Automation Magazine - IEEE ROBOT AUTOMAT*, 19:18–19, 03 2012.

[3] Zhenjun Du, Daokui Qu, Fang Xu, and Dianguo Xu. A hybrid approach for mobile robot path planning in dynamic environments. In *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1058–1063. IEEE, 2007.

[4] Dieter Fox. Kld-sampling: Adaptive particle filters, 2002.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[7] MATLAB. Automated driving toolbox. `https://www.mathworks.com/products/automated-driving.html`.

[8] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.