EECS 467: Autonomous Robotics Laboratory
Reduce Reuse Recycle Robot (RRRobot)

Sravan Balaji*, Chenxi Gu†, Jake Johnson‡, and Derek Witcpalek§
University of Michigan, Ann Arbor
{ *balajsra, †chenxgu, ‡thejakej, §dwitcpa }@umich.edu

Abstract

In this project, we attempt to simulate the classification, manipulation, and sorting of items into trash or recycling. The goal is to explore the challenges involved in separating single stream waste materials into multiple streams, as is done at waste management facilities. Workers who are hired to sort materials in these facilities often leave after a short time, sometimes within hours of being hired. Robots can help fill this role by accurately sorting items and reliably operating for years. We simulate the control of a robotic arm that is able to classify an image, pick up an object, and place it in the correct bin to successfully separate a single stream of waste into a trash and recycling bin. This simulation makes use of a convolutional neural network to classify images, a depth camera to generate point clouds that can be used to determine where the arm should grab, and inverse kinematics to determine a set of joint positions that will allow our robot to reach its desired target pose.

Keywords*: Convolutional Neural Networks, Inverse Kinematics, ROS, Gazebo, ARIAC.*

1. Introduction

In recent years, more and more countries have become aware of the growing issues with waste disposal. Developing countries have begun to ban garbage imports, which in turn has caused both these countries, and countries that once relied on exporting their waste, to reevaluate the their waste disposal methods. A significant step in garbage processing is garbage classification. Recyclable and trash are separated such that materials like paper, glass, plastic can be reused. While it has been partially done automatically at recycling plants, this work is still quite labor intensive.

In this project, we aim to simulate the scenario at a recycling facility where a conveyor belt of items needs to be separated into trash and recycling. This project contains three major modules: detection, classification and control. Assume an RGB-D camera is installed on top of the conveyor belt which provides color information of an area as well as depth information. When an object is moving on the conveyor belt, our system will first detect it using the point cloud calculated from the depth information, then classify it using a computer vision model and finally control a robot arm to grasp the object into the recyclable / trash bin. Our simulated scenario is similar to this, with a few modifications for simplicity.

This report is organized as follows: In Section 2, we describe the high-level architecture of the RRRobot system, how the three major modules communicate with each other and the software environment where we develop this system. In Section 3, we discuss how to train a classifier which separates different materials based on vision information and its performance. In Section 4, we discuss how to use the point cloud data from a depth camera to detect the existence of and estimate a bounding box around an object. In Section 5, we discuss how to control the end effector of the robot arm to move towards a desired location with inverse kinematics. In Section 6, we conclude what we've learned and achieved from this project. Source code, documentation, and videos can be can be found on the project website.

2. System Architecture

In this section, we first describe the docker environment we used for easier collaboration and the GEAR environment where we developed our system. Then we introduce the high-level architecture of our system and how different modules communicate with each other.

2.1. Development Environment

Our simulation was implemented in a docker container [1] running Ubuntu 18.04.4 LTS with ROS Melodic Morenia [2], Gazebo 9.0.0 [3], and Python 3.6.9 [4]. We used docker to ensure a consistent development environment as well as to ensure that others can run our simulation without worrying about installing dependencies manually. Instructions for running the simulation environment can be found on the project website.

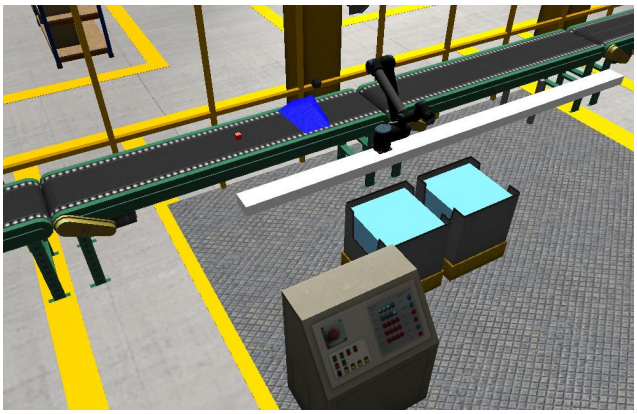


Fig. 1: RRRobot Modified Gazebo Environment for Agile Robotics 2019 [5]

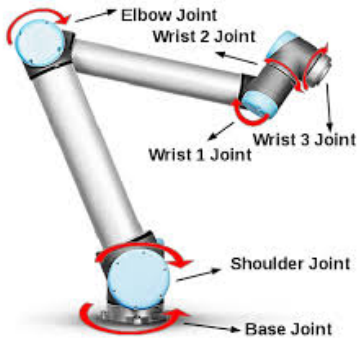


Fig. 2: Universal Robots UR10 Robotic Arm Degrees-of-Freedom, reproduced from [6]

2.2. GEAR

To simulate our project, we used the Gazebo Environment for Agile Robotics (GEAR) [5] shown in Figure 1. GEAR is a simulation environment used in the Agile Robotics for Industrial Automation Competition (ARIAC) 2019 [7]. We chose to use this environment because it contains all of the components that our project needed. The GEAR environment is modeled after an industrial setting and includes a conveyor belt, multiple sensors for detecting and identifying objects, and robotic arms for manipulating the objects.

We had initially planned on custom making CAD models that would be imported into our Gazebo environment, but we found that the time required to implement that would not improve our end product as compared to what was already publicly available. Switching to using GEAR allowed us to focus on implementing the object classification, object detection, and motion planning components of the project rather than environmental setup.

The conveyor belt is used to move objects into position so that they can be detected by the depth camera and picked up by the arm. The main node sets the conveyor belt power to 0% when it receives a target grasping pose from the depth camera node. It sets the conveyor belt power to 100% again when the arm controller releases an object. We used two of the bins provided in the GEAR [5] environment to sort recyclable and trash items. We positioned the depth camera above the conveyor belt at an angle in order to capture multiple surfaces of the objects. Positioning the camera directly above the conveyor belt would have likely sufficed for detecting the cubes used in our simulation, but the angled position is what we would use for more complicated object shapes. The arm used in our simulation is the UR10 industrial robot arm from Universal Robots [8]. The UR10 (shown in Figure 2) is a 6 DOF arm with a radius of 1.3 meters. The base of the arm is attached to a linear actuator which allows the arm to move freely parallel to the conveyor belt.

We decided to use simple color coded cubes to represent the items that come down the conveyor belt. Green indicates recyclable and red indicates trash. We had originally planned on using realistic CAD models of a plastic bottle, cardboard box, etc. However, we found that using these models made the gripping process much more challenging, so we instead opted for simple cubes. Internally, the CV model is being fed images, such as those seen in Figure 4, that it uses to classify the items, but the arm only needs to pick up simple cubes rather than objects with curved surfaces at varying orientations.

2.3. ROS Nodes Communication

A flow diagram of the interaction between the simulation environment and our ROS nodes is shown in Figure 3. Five ROS nodes are used to create, detect and classify objects, and move the arm to grab items off the conveyor belt. When the simulation starts, an object is spawned on the conveyor by the object spawner node. This node publishes a message with the image name for the CV model to classify. Once the object reaches the depth-sensing camera, the depth camera ROS node publishes a desired grasp pose, and the conveyor belt is stopped. The final piece is the arm controller, which uses the pickup location from the depth camera node and the classification from the CV model to move the arm and correctly sort the item.

The main node (labeled RRRobot in Figure 3) ties together all these pieces. It controls the flow of the simulation by starting and stopping the conveyor belt, and setting destinations for the arm as other nodes provide information about the world. The RRRobot node listens for the item pickup location and whether it is trash or recyclable. Once this information is received, the node tells the arm controller where to move the arm. It also controls the conveyor belt. This involves stopping the belt when an object is in view, and starting the conveyor back up when an object is dropped in the bins.

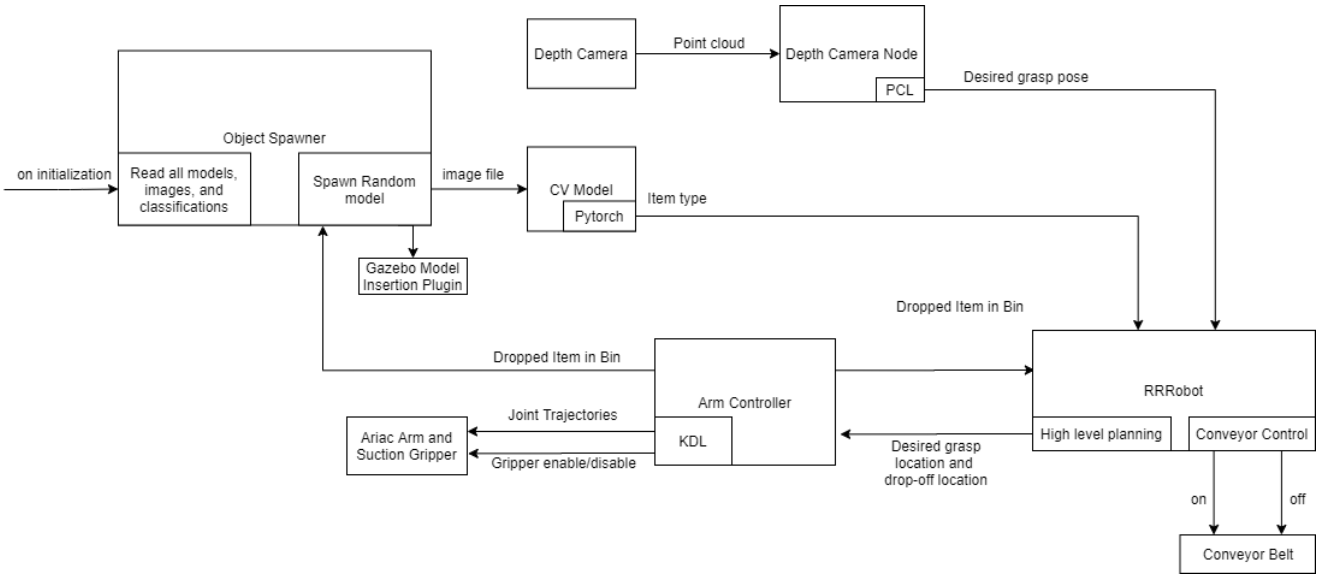


Fig. 3: RRRobot Simulation Architecture Model

3. Garbage Classifier[1]

In the early days, automatic garbage classification made use of the physical properties of materials. For example, different materials have different densities. When objects move along a conveyor belt in a recycling plant, with wind blowing up from the bottom, the lighter objects such as plastic bags and papers will float into the air while the heavier objects like metal will stay on the conveyor belt. This is a simple way to separate objects of different densities. This method is so effective that recycling plants are still using it today [9].

It wasn't until recently that people started to use computer vision techniques to help garbage classification. "Auto-trash", [10] built in 2016, is the first garbage classification project to use computer vision, to our knowledge. When an object is place upon the top of this container, it will use the camera that it is equipped with to sort the object into recyclable or compost, and then rotate its top such that the object falls into the correct half of the container.

In addition, in 2016, Mindy Yang and Gary Thung [11] from Stanford collected their own dataset and made it public. Their dataset consists of images of cardboard, glass, metal, paper, plastic, and trash, which are targeted for garbage classification because the first 5 categories in this list are the main categories used for recycling. They also experimented with two types of machine learning algorithms, SVM and CNN, for image classification. For their SVM, the feature vector of an image is extracted using the SIFT algorithm; for their CNN, its architecture is similar to AlexNet. Their SVM model achieved a test accuracy of 63%, while their CNN model only achieved a test accuracy of 22%.

In 2018, Cenk Bircanoglu et.al [12] used the dataset built by Mindy Yang and Gary Thung and experimented with different data augmentation strategies, model architectures, and optimizers. They found that fine-tuning a 121 layered DenseNet allowed it to achieve a test accuracy of 95%. They also proposed a new architecture called "RecycleNet", which could achieve a test accuracy of 81% when learning from scratch.

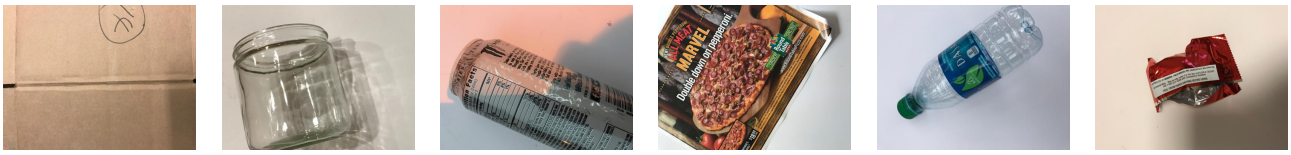
Inspired by these previous work, we built a garbage classifier with computer vision techniques.

3.1. Train classifiers

We used the dataset built by Gary Thung and Mindy Yang from Stanford University in their project [11]. This dataset consists of 393 images of cardboard, 491 images of glass, 400 images of metal, 584 images of paper, 472 images of plastic, and 127 images of trash. Figure 4 shows some example images from this dataset. Each image is in RGB format with a width of 512 pixels and a height of 384 pixels. We split the dataset such that the amount of training, validation, and test images have a ratio of 70:15:15 in accordance with each type of waste. In addition, we augmented the training data by flipping each image horizontally and vertically to get two additional images. We ended up with 5298 images in the training dataset, 398 images in the validation dataset, and 398 images in the test dataset.

We adapted 3 architectures from MobileNet, ResNet and InceptionNet. The last few layers for each one were modified such that the model was also able predict and output the log probability that an image belonged to each of the six

1. Part of the work in this section comes from the previous work of Chenxi (one of the authors of the report) with Yifan Yang (yifany@umich.edu) and Shaayan Syed (sshaayan@umich.edu) in another course project.



(a) cardboard (b) glass (c) metal (d) paper (e) plastic (f) trash

Fig. 4: Example images in Stanford dataset [11]

categories. These architectures are detailed in our github repo. Since the models trained on ImageNet are available to the public, we were able to experiment with the three following frameworks:

- Pretrain-Finetune: Load the original model trained on ImageNet and update all the layers on our training data;
- Pretrain-Freeze: Load the original model trained on ImageNet. Freeze layers from the original model and only update the layers we modified on our training data;
- Learn from scratch: Learn all the layers on our training data

We experimented with 3 different model architectures, 3 different frameworks and 4 different learning rates {1e-3, 5e-4, 1e-4, 5e-5}, and ended up with 36 models. We trained each model for 50 epochs. Our highest test accuracy was 95% and was achieved by the modified MobileNet and the modified InceptionNet, both of which had a learning rate of 5e-5 using the pretrain-finetune framework. Since querying the modified MobileNet is faster than querying the modified InceptionNet, we choose the modified MobileNet as our final classifier.

3.2. Evaluate our best classifier in detail

The confusion matrix in Figure 5 details the performance of our final classifier on the test data set. The labels in the vertical direction represent ground truth, and the labels in the horizontal direction represent predictions. The numbers placed along the diagonal line in the middle indicate wrong predictions.

We further analyzed the wrong predictions. Out of 20 wrong predictions, 4 images had wrong ground truth (they were somehow marked with wrong labels by the people who built this dataset), 10 images were "tricky", and 6 images were "manageable". By "tricky", we mean that it could be a little challenging, even for a human being, to correctly classify the object in the image, either because of the angle at which the photo was taken, the way the object was illuminated, etc. Some tricky cases are shown in Figure 6. And by "manageable", we mean that it should be easy for the average person to correctly classify the object in the image. Some manageable cases are shown in Figure 7.

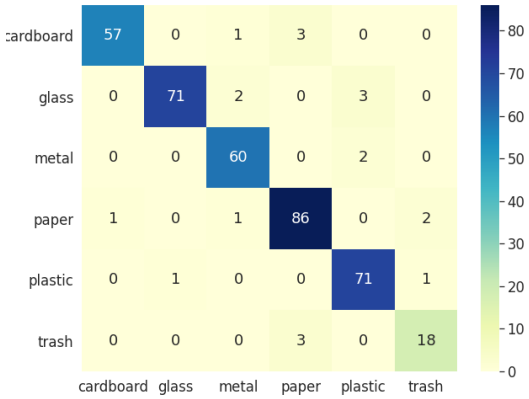
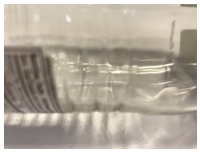


Fig. 5: Confusion matrix on test data from Stanford dataset [11]

(a) The cardboard is misclassified as paper, perhaps because of its texture and coloring

(b) The glass is misclassified as metal. This may be due to its reflection



(c) The plastic is misclassified as glass, perhaps because of its reflection as well

(d) The paper is misclassfied as trash, which could be because its size is close to that of a snack pack

Fig. 6: Tricky cases

3.3. Future work to improve the classifier

One approach which could be essential to improve our classifier is to collect a large and diverse garbage dataset. There are several issues we noticed in the Stanford dataset.

(a) The trash is misclassified as paper

(b) The paper is misclassified as cardboard

Fig. 7: Manageable cases

First, most of the images in this dataset had plain, white backgrounds. Second, the Stanford dataset is not diverse when it comes to the types of objects it has images of. For example, the "glass" images in the training dataset contain only bottles and the "metal" images contain either metal bottles or cans. However, in a real testing scenario, the background color could be different and "metal" could contain silverware or metal tools. This means the performance of our classifier could drop a lot when applied to a recycle plant. If we can construct a more diverse dataset in a more extensive way, we will probably obtain better results on real-world tests.

We should also notice that in real world, it is sometimes difficult for a human being to correctly classify an object only based on vision. Instead, human beings usually also rely on tactile sense. From our experiments, we find that plastic objects may be confused with glass and that paper may be confused with trash. This indicates that we might need to combine the sensation of touch along with vision in order to build a better waste sorter. Researchers from MIT [13] have already done something to this extent by creating artificial fingers with tactile sensors for robots. The robots were then able to detect whether an object they touched was either paper, metal, or plastic based on its size and stiffness. They achieved a test accuracy of 85%. Although this accuracy is not high, it might help us solve the problem of distinguishing different types of garbage, such as glass and plastic. A plastic bottle will undergo a significant change in shape if you put pressure on it, while a glass bottle will not. The vision model and the touching model could complement each other.

Right now, our classifier works on the image when there is only one object to detect. It would be great if it could classify each object in an image containing multiple objects. To do this, we would probably want to train a faster R-CNN model which predicts a bounding box and then classifies each object in the image that has a bounding box. We did not implement this feature in this project because the dataset we used did not contain ground truth about the bounding box. This could be the future work for a classifier in a real recycle plant.

4. Object Detection

4.1. Depth Camera

Our simulation environment uses a depth camera to generate point cloud data. This is used to detect when an object is in position on the conveyor belt and to determine where the arm should grasp the object. To work with the point cloud data, we used the Point Cloud Library (PCL) [14], which is an open source library for image and point cloud processing. PCL provides a method for segmenting a point cloud into multiple clusters which can be used to identify different objects within a single point cloud. We used this method to extract the points corresponding to the object from the rest of the point cloud.

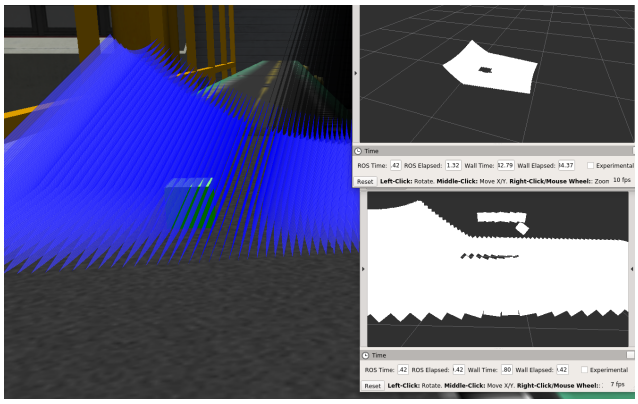


Fig. 8: Point Cloud collected by Depth Camera

Figure 8 shows the cube on the conveyor belt and the point cloud captured by the depth camera at two different angles. As shown in the bottom right point cloud image, the top surface of the cube is clearly visible as the measured depths on this surface are smaller than the depths measured around it, indicating that an object is in frame.

Point cloud data collected from the depth camera must first be transformed from the camera's frame into the world frame. This is done using the ROS transformations package and the PCL-ROS package, which allows us to easily transform a point cloud using a transformation defined by a point representing the position of the frame and a quaternion representing the orientation of the frame. To detect if an object was in position on the conveyor belt we approximated the position of each cluster extracted from the point cloud and checked if any of the clusters was located within a predetermined area. Once the object is detected and the conveyor belt is stopped, the depth camera node estimates a 3-dimensional bounding box for the object using the maximum and minimum x, y and z coordinate values in the point cloud as well as the known height of the conveyor belt. Our simulation uses highly simplified representations of objects, so this approach accurately captures the pose of the object on the conveyor belt. The end effector of the arm uses a vacuum gripper, so the target pose is directly against the surface of the object. The depth camera node publishes the target pose so that the main node can send a command to the arm controller.

4.2. Future work for improved object detection

This simple object localization method works well for the cubes used in our simulation, but for more complicated

shapes a more sophisticated method would be necessary. PCL includes methods for computing surface normals of point clouds as well as for cylindrical segmentation, which could be used to find the surfaces of objects of various shapes. Our simulation also has only a single object on the conveyor belt at a time, whereas a realistic recycling plant scenario would likely have many objects clumped together. Extending our current project to work with various object shapes and multiple objects grouped together would be the next steps to take in making this project applicable in real-world recyclable sorting environments.

5. Arm Controller

The arm controller is given a pose to grab the object at, and the location of the bin to drop the object in. It uses inverse kinematics to find the joint positions required to reach these locations. We also added two intermediate locations in the arm's path so that it avoids all obstacles. In our environment, there are no objects above about 1 meter, so if we keep the arm above 1 meter while moving, collisions are very unlikely. This was a compromise since we weren't able to get full path-planning with obstacle avoidance working in MoveIt.

5.1. MoveIt

As suggested in the GEAR wiki, we initially interfaced with MoveIt [15] via RViz [16]. We were able to use the GUI to plan and execute motion planning to a desired position, while taking into account obstacles. However, when we attempted to use the C++ MoveIt interface, we encountered many issues with the provided MoveIt configuration files for the UR10 robot arm. We found that the linear actuator link that allows the arm to slide along the rail was being dynamically added to the model when the simulation started, so we were unable to find a way to programmatically use MoveIt for path planning. Our decision to abandon MoveIt proved to be the best option when we found that others were having similar issues, but the ARIAC support team would not provide technical support for MoveIt or the configuration files.

5.2. Kinematics and Dynamics Library

After many issues trying to get the UR10 arm working with the C++ MoveIt interface, we decided to switch to using the Orocos Kinematics and Dynamics Library (KDL) [17] for forward and inverse kinematics. We had experimented with it prior to using the ARIAC environment, so the transition wasn't too difficult.

Our arm controller used the KDL Levenberg-Marquardt inverse kinematics solver. This is an inverse kinematics solver that starts from an initial guess pose and uses iterative gradient-based optimization to find the robot configuration that will produce the desired end effector pose. This method converges more quickly than others if the initial guess is close to a correct configuration, but doesn't handle arbitrary initial guesses well [18]. It can be prone to getting stuck in local minima, and works best when calculating the configurations for a series of poses along a trajectory. We knew the configuration of the previous point, which was either above the conveyor belt, or above the bins, so this seemed like a reasonable method to try.

When using inverse kinematics to solve for the joint positions required to reach the desired pose, we had issues with the solver getting stuck in local minima. At first, we tried increasing the maximum number of iterations used by the solver, but it still returned errors. We also thought that changing the tolerated error in the final position and for the individual joints would help, but because the planner was getting stuck in a local minimum, these changes didn't make a difference. Iterative solvers commonly have problems with getting stuck in local minima. The best solution we found was to adjust the initial conditions by a random amount in order to help the arm avoid these minima. If the state guess we pass into the inverse kinematics solver is far enough from the local minimum, a valid solution is found.

In our final solution, the arm controller tries to calculate the joint positions to reach the destination using the current configuration as an initial "guess" for the inverse kinematics solver. If this fails, it tries again with +/-0.1 radian adjustments randomly applied to each of the joints. This process repeats up to a maximum number of attempts and then the arm controller gives up on reaching that point, assuming that it is outside of the robot configuration space.

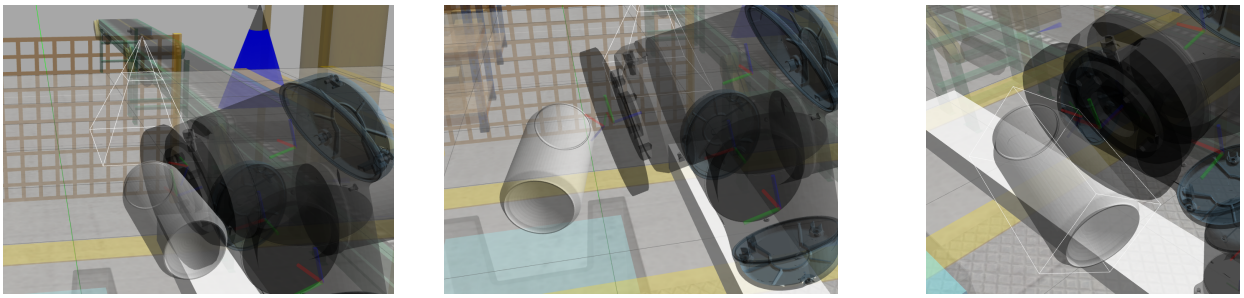
5.3. Gripper

We used the vacuum gripper plugin in Gazebo. This end effector allows the robot to create fixed links to dynamically attach to, and manipulate objects in the environment. In initial testing of the plugin, we noticed that it was very temperamental, requiring the end effector link frame to be well lined up with the frame of the object. As you can see in Figure 9, some gripper poses attached to the object while very similar poses didn't. The gripper seemed to have better performance when grabbing an object off of a fixed surface than trying to attach the gripper to a floating object.

In order to ensure the gripper attached to the desired object, we first moved the arm end effector to a position slightly above the object. The gripper was slowly lowered until the end effector attached to the object. This ensured that objects weren't tipped over or otherwise disturbed while we grabbed them.

5.4. Future Work

If we had additional time to work on this project, there are a couple of key points that could use improvement. The main issue that comes to mind is motion planning with obstacle avoidance. Currently, KDL does not take into account the presence of other objects in the environment when determining the joint positions that will achieve the desired end effector pose. MoveIt does provide this capability, but as mentioned, we were unable to get that working. As shown in our project's



(a) able to attach (b) not able to attach (c) not able to attach

Fig. 9: The gripper needed to be well-aligned to grab objects

video, the arm hits the depth camera that is above the conveyor belt, but eventually manages to get past it when picking up the cubes. We could get around this collision issue by specifying different intermediate points in the robot arm trajectory that are out of the way of the camera, or we can simply disable collisions on the camera model in Gazebo. Additionally, we could have improved the intermediate poses used in motion planning. As shown in the project video, the joint position calculated by inverse kinematics often requires some joints to rotate nearly 360. If we choose the intermediate poses more carefully, perhaps IK would rotate the joints fewer degrees in the opposite direction to achieve the same position.

We would have also liked to improve the gripping capability of the robot arm. As shown in Figure 9, the gripper requires very close alignment with objects such as the soda can. We decided to use cubes to represent the items as this had a much higher likelihood of success, but it would have been interesting to explore how the suction end effector could work for objects with curved surfaces and facing different orientations (e.g. vertically vs horizontally placed on conveyor belt).

6. Conclusion

In this project, we simulated the scenario at a recycling plant where a conveyor belt of items needs to be separated into trash and recycling. We address this problem with three major modules: detection, classification and control. The detection module detects the existence of an object in the designated area on the conveyor belt; the classification module, which uses convolutional neural networks, classifies the object into trash or recycling. The control module manipulates the robot arm to grasp the object and then puts it into the correct bin. Through our simulation based approach to our implementation, we gained hands-on experience about the methods used in classification of images, control of robot arms, and manipulation of objects in 3D space. We also gained experience with ROS, Gazebo, and KDL that will aid us in our future work. Although this project was ultimately done in a simulation environment, instead of our initial plan to use a hardware-based approach on a real robot, the experience of implementing this project is still useful for building a real robot system in the future.

7. Acknowledgments

We would like to recognize the course staff of EECS 467 for their help this semester. Professor Chad Jenkins (ocj@umich.edu), Graduate Student Instructor Xiaotong Chen (cxt@umich.edu), and Graduate Student Instructor Jana Pavlasek (pavlasek@umich.edu) were instrumental in helping us learn many robotics concepts through lectures and hands-on lab projects. Due to COVID-19, Our project transitioned from a heavily hardware based to a heavily software and simulation based approach once classes became remote. With the help of the course staff, we were able to make the best of an unfortunate situation.

Additionally, we would like to recognize our fellow classmates in EECS 467 who voluntarily offered their help when students had questions about lecture, labs, projects, and anything else. Without them, this project and course as a whole would not have been as fun and engaging as it was.

References

[1] "Docker." [Online]. Available: https://www.docker.com/
[2] OSRF, "Robot operating system." [Online]. Available: https://www.ros.org/
[3] ——, "Gazebo." [Online]. Available: https://www.gazebosim.org/
[4] "Python." [Online]. Available: https://www.python.org/
[5] OSRF, "Gazebo environment for agile robotics." [Online]. Available: http://gazebosim.org/ariac
[6] A. Topalidou-Kyniazopoulou, "Motion planning strategy for a 6-dofs robotic arm in a controlled environment." [Online]. Available: https://www.ais.uni-bonn.de/theses/Angeliki_Topalidou-Kyniazopoulou_Master_Thesis_07_2017.pdf
[7] NIST, "Agile robotics for industrial automation competition," Apr 2020. [Online]. Available: https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition
[8] "Collaborative robotic automation: Cobots from universal robots." [Online]. Available: https://www.universal-robots.com/
[9] L. Ioannou and M. Petrova, "America is drowning in garbage. now robots are being put on duty to help solve the recycling crisis," Jul 2019. [Online]. Available: https://www.cnbc.com/2019/07/26/meet-the-robots-being-used-to-help-solve-americas-recycling-crisis.html

[10] J. Donovan, "Auto-trash sorts garbage automatically at the techcrunch disrupt hackathon," Sep 2016. [Online]. Available: https://techcrunch.com/2016/09/13/auto-trash-sorts-garbage-automatically-at-the-techcrunch-disrupt-hackathon/
[11] M. Yang and G. Thung, "Classification of trash for recyclability status," *CS229 Project Report*, vol. 2016, 2016.
[12] C. Bircanoğlu, M. Atay, F. Beşer, Ö. Genç, and M. A. Kızrak, "Recyclenet: Intelligent waste sorting using deep neural networks," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, 2018, pp. 1–7.
[13] A. Conner-Simons, "Robots that can sort recycling," Apr 2019. [Online]. Available: http://news.mit.edu/2019/mit-robots-can-sort-recycling-0416
[14] "Pcl - point cloud library (pcl)." [Online]. Available: http://pointclouds.org/
[15] PickNik, "Moveit motion planning framework." [Online]. Available: https://moveit.ros.org/
[16] D. Hershberger, D. Gossow, and J. Faust, "Rviz." [Online]. Available: http://wiki.ros.org/rviz
[17] Orocos, "Orocos kinematics and dynamics." [Online]. Available: https://www.orocos.org/kdl
[18] "Inverse kinematics algorithms." [Online]. Available: https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html