RecycleBot

Meghana Keeta, Serena Ying, Mae Zhang
Department of Electrical and Computer Engineering, Carnegie Mellon University
{mkeeta, sjying, maez}@andrew.cmu.edu

Abstract

Plastic waste has been a major focus of environmental awareness in recent years. In order to combat this issue, we have designed a RecycleBot to autonomously detect and collect plastic bottles into its storage for later recycling. RecycleBot uses computer vision and machine learning to efficiently detect, navigate to, and collect plastic bottles into its storage for later recycling. It should be able to collect plastic bottles of various shapes, orientations, and sizes and also avoid obstacles in its path. RecycleBot not only removes the need for people to manually pick up littered bottles, but also improves the rate of recycling such items, contributing to the effort to preserve our planet.

Index Terms

Index Terms— **OpenCV, Computer Vision Object Tracking, Image Processing, iRobot Create 2, LiDAR Depth Sensing, Machine Learning for Object Detection, Transfer Learning, Jetson Xavier NX, Recycling**

I. Introduction

22 billion plastic water bottles get thrown away rather than recycled every year in the US [1]. Furthermore, more than 60 million plastic water bottles are thrown away each day, many of which end up as litter in streets, parks, and waterways [2]. It is evident that plastic bottles are a critical source of litter in the US, which means that the endeavor to reduce the litter in public areas is high effort in terms of resources and manpower.

That's where RecycleBot comes in— using a trained machine learning model combined with computer vision to detect plastic water bottles, it is an iRobot-based system that provides a solution to plastic waste in our environment. RecycleBot autonomously travels within an area to systematically scan for and detect plastic water bottles in an environment with litter, collecting the bottles in its internal storage. Users of RecycleBot will benefit twofold from its autonomous bottle collection functionality. On one hand, it removes the need for human labor to reduce the amount of plastic litter, and on the other, automates the process of sorting between recyclable plastic water bottles and other types of non-water bottle and non-recyclable waste. Hence, this solution will not only reduce the amount of resources dedicated towards managing plastic bottle litter, but also improve the rate of recycling such items, for both economic and environmental gain.

Research into competing and adjacent technologies reveals that automated recycling systems are available, however, many aim to automate the sorting process between recycling and trash [3], which still requires humans to deposit their trash into designated areas. Other mobile cleaning robot solutions include those meant to collect litter in local waterways [4] or litter on beaches [5]. Beyond the location difference between such robots and RecycleBot, which is built to operate indoors or on outdoor hard, concrete surfaces, these robots are more focused on collecting trash rather than collecting only recyclable items which can all go into recycling without additional sorting.

II. Use-Case Requirements

To ensure RecycleBot meets its intended goal to efficiently detect and collect plastic bottles for recycling, as well as to reflect the updates we have made throughout the duration of this project, we have the following final use case requirements.

RecycleBot must be fully autonomous and operate indoors on smooth, hard terrain. The RecycleBot being fully autonomous is crucial to its success, as we are providing a solution to reduce the manpower required to clean up public spaces, many of which are concrete or paved. Our design is meant for cleaning public places with hard terrain because it is the most feasible and useful area of operation.

In order to collect bottles successfully, they will first need to be identified in the environment. RecycleBot's bottle detection model **must correctly detect the following items** at the listed accuracies, with **less than a 10% false positive rate** for each:

1) **Standard size empty plastic water bottles at different orientations and crush levels with 90% success rate.**
2) **Obstacles with 80% success rate.**

Differing from our design report, we have redefined our use case requirements to focus on all empty plastic bottles, as we found that our bottle detection algorithm had more than sufficient tolerance for different plastic bottle orientations and states that it was most reasonable to consider all such bottles as a single category. Obstacles are defined as anything in the environment within a 1 meter radius of RecycleBot that is not a bottle, specifically other commonly littered items such as aluminum cans and plastic drink quarts, as well as glass bottles, the three other items in our training dataset. These classification accuracies will be tested in the real world environment where there will be items scattered around the RecycleBot within a 1 meter radius. **We chose 90% accuracy for our bottle detection** because the detection model may misclassify on instances where the bottle is too far away or obscured by another item. **We chose 80% accuracy for our obstacle detection** as the detection model may have difficulty distinguishing between items which are a part of our trained dataset and items which are just in RecycleBot's surroundings in our testing location in TechSpark.

Next, we require RecycleBot to **avoid obstacles with a 80% success rate**. In the real world, we expect items that are not bottles in the environment, so RecycleBot needs to be able to successfully avoid bumping into or picking up obstacles that are in its path. We have allowed for a margin of error due to the fact that the robot may have complications picking up accurate distance readings while operating at an efficient speed. To account for the 20% of unavoided obstacles, RecycleBot's construction allows it to avoid accidentally intaking items which do not match the approximate height of our plastic water bottles, making our obstacle avoidance success rate viable.

To confirm successful navigation and hardware mechanics of the system, we require RecycleBot to **pick up and store detected plastic water bottles with a 70% success rate**. This tests the RecycleBot's ability to successfully navigate to a detected bottle and activate its intake to collect the object into its internal storage area. We have allowed for a margin of error due to the various sizes, shapes, and orientations of bottles the intake is expected to pick up.

Finally, we require RecycleBot to be timely and efficient. RecycleBot is required to take **on average less than 1 minute and 15 seconds to identify all 3 bottles distributed within a 1 meter radius and attempt to pick up as many as 3 of those bottles** with no obstacles present. To elaborate on the quantitative details of this task, we referenced a real world use case of RecycleBot to determine both the timing and the search area. As RecycleBot is expected to be used on smooth concrete, we chose a basketball court to be our test field, given their relative ubiquity in public parks and likelihood to be littered with plastic water bottles and other types of waste. For our search area, since we want RecycleBot to search in circular areas each run, eventually covering the entirety of the basketball court, we had to consider the way that RecycleBot would be able to navigate to a new point to center its search area in between runs. The most logical way would be to utilize GPS to pinpoint RecycleBot's location and to calculate the location for the new center point. Through research into GPS accuracy limitations, we found that in practice, the best case horizontal error of GPS assisted by the Wide Area Augmentation System (WAAS) is 1 meter [6]. However, to account for outdoor conditions where there is less visible sky and thus a reduced accuracy of received location data from the WAAS system of satellites, we assume a worst case horizontal error of 2 meters. Thus, we want our robot to see a maximum of 2 meters in any direction, which is why we chose a search radius for this test of 1 meter. For our timing requirement, we want RecycleBot to be able to clear an entire basketball court in under two hours in order to avoid extreme light conditions changes such as the sun setting affecting the visibility of the robot, since many public parks in the US close near sunset and would likely run RecycleBot near closing time. With a search radius of 1 meter, RecycleBot would need to complete around 100 iterations to cover a basketball court. To complete its searches in under two hours, it would need to finish each run in 72 seconds on average. We have rounded this value up to 75 seconds, or 1 minute and 15 seconds, since we don't expect to see such a high density of water bottles on the court (3 bottles in 1 m radius).

III. Architecture and/or Principle of Operation

Our system relies on computer vision and our tailored machine learning model to detect and locate plastic bottles and obstacles, then utilizes the LiDAR camera's depth information to send navigation commands to the iRobot and intake mechanism. These system components, divided into our software system and our hardware system, are outlined in Figure 1.

A. *Software System*: Our software pipeline, found in Figure 2, defines RecycleBot's functionality. From the environment, the RealSense LiDAR Camera sends a color stream and a depth stream to the Jetson Xavier NX, powered by a 12V Battery situated at the back of the structure behind the iRobot, over USB to facilitate the software pipeline, the heart of RecycleBot. The software pipeline involves first performing inference with our trained YOLOv5 machine learning model, which generates a set of bounding boxes of bottles and obstacles that are used in target selection. Once a target has been selected based on the depth stream as well as relative positioning, OpenCV object tracking is used to begin and support our navigation calculations. The navigation calculations include calculating the duration and direction for the iRobot to turn to center the target to the robot, and tracking the distance between the robot and the target as well as the distance between the robot and any close obstacles that need to be avoided. These navigation commands are sent to the iRobot over UART through a Python wrapper library [18] and the Arduino Uno through USB which controls the Cytron MD30C motor driver for the intake mechanism once RecycleBot moves close enough to the target to pick it up.

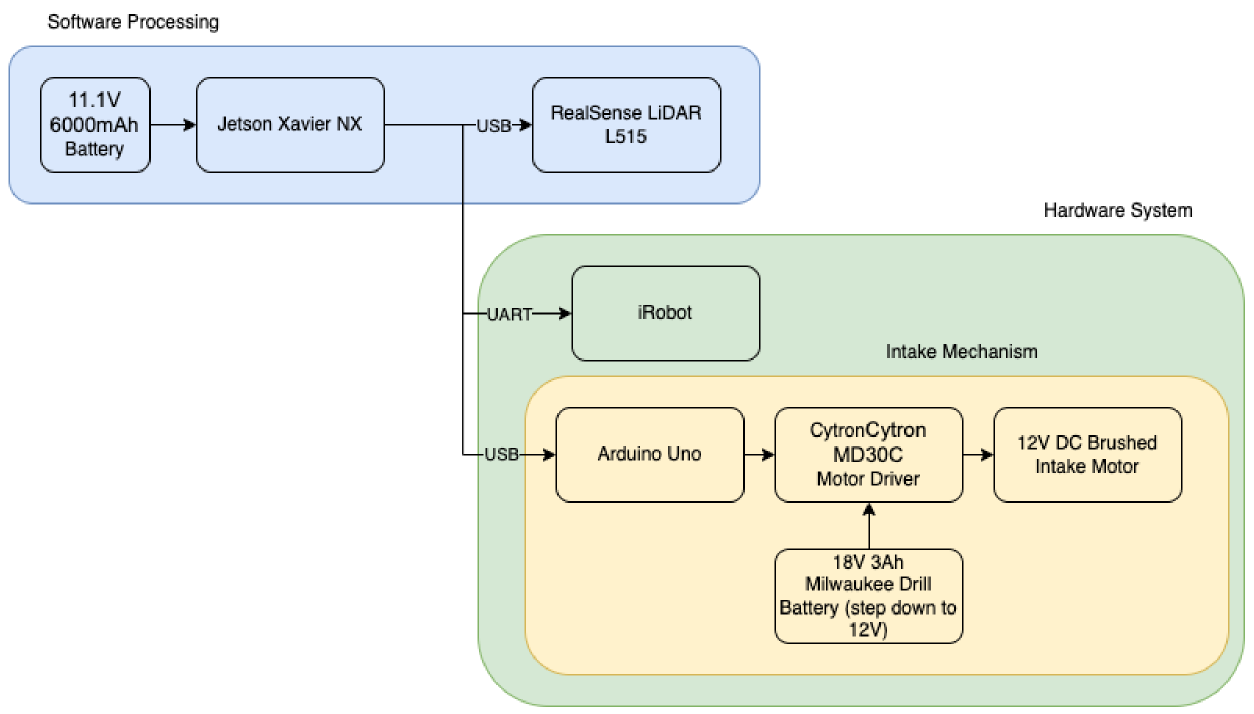


Fig. 1. System Block Diagram.

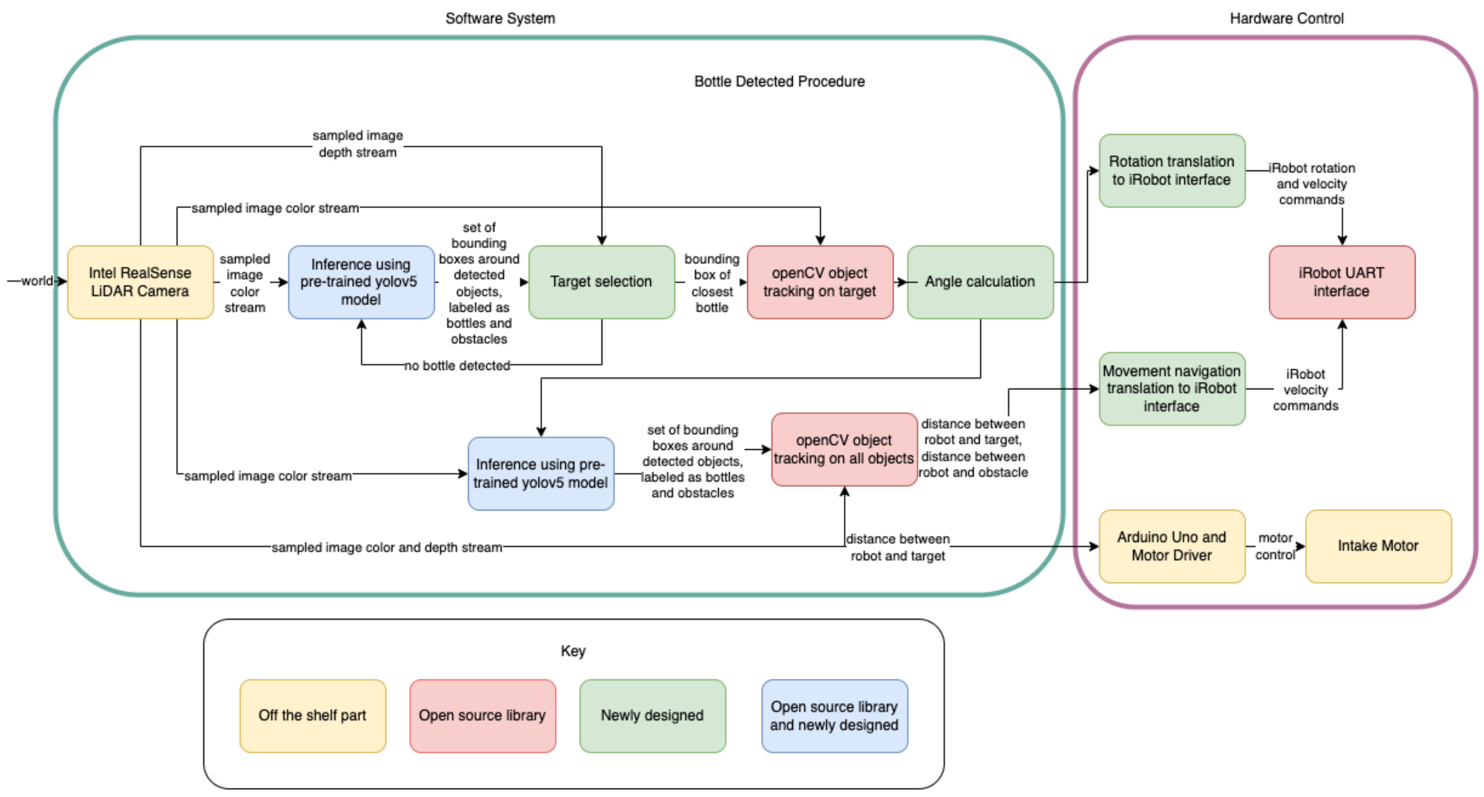

Software System
Hardware Control
Bottle Detected Procedure
world
Intel RealSense LiDAR Camera
sampled image depth stream
sampled image color stream
sampled image color stream
Inference using pre-trained yolov5 model
set of bounding boxes around detected objects, labeled as bottles and obstacles
Target selection
no bottle detected
bounding box of closest bottle
openCV object tracking on target
Angle calculation
Inference using pre-trained yolov5 model
set of bounding boxes around detected objects, labeled as bottles and obstacles
openCV object tracking on all objects
distance between robot and target, distance between robot and obstacle
sampled image color stream
sampled image color and depth stream
distance between robot and target
Rotation translation to iRobot interface
iRobot rotation and velocity commands
iRobot UART interface
Movement navigation translation to iRobot interface
iRobot velocity commands
Arduino Uno and Motor Driver
motor control
Intake Motor
Key
Off the shelf part
Open source library
Newly designed
Open source library and newly designed

Fig. 2. Software Pipeline.

B. *Hardware System*: The hardware and robot components are shown in Figure 3. On top of the second lexan sheet which sits across the arms of the structure, above the intake shaft sit all our control elements— the Jetson Xavier NX, Arduino Uno, Cytron MD30C, the Intel RealSense LiDAR Camera, which sits at an angle on a custom 3D printed round mount, and all the associated wiring. The chassis of our robot that allows the robot to move is an iRobot, commanded by the Nvidia Jetson NX over UART. The iRobot only has top mounting through the nameplate, so the rest of the structure is supported by the wood beam in the center. The center wood beam is connected to 2 side wood beams which lean against the sides of the iRobot and are supported by caster wheels mounted to the bottoms of the beams on the intake side. Connected to the center wood beam and the side wood beams is a lexan sheet that lays flush to the top of the iRobot which behaves as the bottom of the water bottle storage area. In the front of the robot, the intake is mounted, driven by a 12V DC Brushed Motor that is attached to the right side of the iRobot and powered by a Milwaukee 18V Power Drill Battery situated at the back of the structure behind the iRobot, stepped down using an 18V to 12V converter. Once the software system generates the navigation calculations to center the iRobot at the target, it passes the rotation duration to the iRobot over UART and utilizes the distance readings from the Intel RealSense LiDAR camera to determine the movement velocity to the iRobot. Those distance readings are also used to signal the Arduino Uno to command the Cytron MD30C motor driver to start and spin the intake motor to pick up the target bottle.

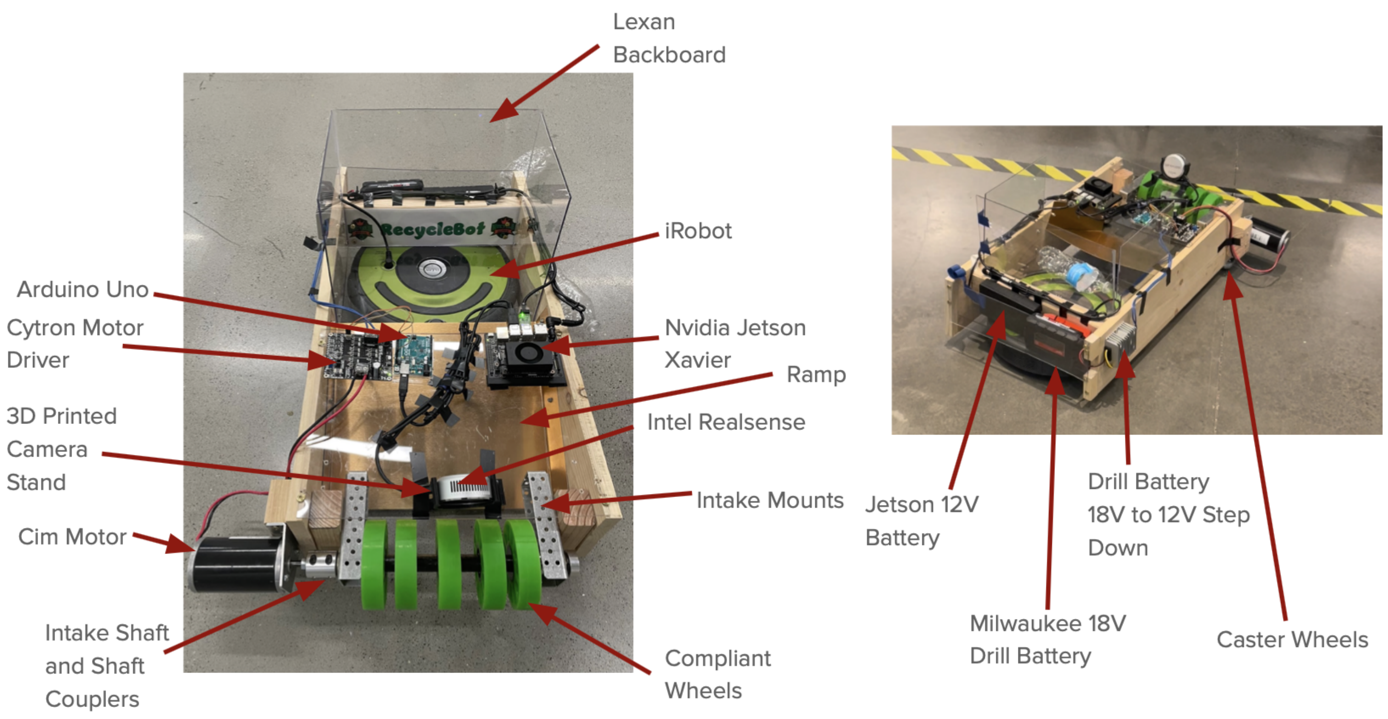


Fig. 3. System Block Diagram.

IV. Design Requirements

Our design requirements define the expected technical behaviors of our system and relate to our use case requirements. In testing our product, we have scoped our project to the two aforementioned cases of 3 plastic bottles within a 1 meter radius of the robot, focusing on the case without obstacles.

For our software system, we impose a requirement for the FPS process rate of the image stream from the LiDAR camera. Through testing, we know that the bottleneck in the software system is running inference on the sampled image stream, so we require our system's **inference to take approximately 1 second per run** in order to minimize delays in RecycleBot's movement. This means our trained ML model takes on average 1 second to output bounding boxes around detected objects every time inference is run. Since this rate limits the amount of information the robot has about its environment if solely reading from inference, we use an OpenCV object tracking algorithm on the 60 FPS color stream from the LiDAR camera in order to bridge the frame gaps in between each frame we sample for our trained ML model. As the OpenCV tracking algorithm we are using works well in real time, we plan to utilize the tradeoff between processing rate and number of tracked objects in order to track all the detected objects in the robot's field of view, allowing us to not only maintain accuracy when traveling towards a bottle but also to detect and avoid obstacles. Including the supplemental use of a tracking algorithm to follow the labeled bounding boxes given by our model in between allows RecycleBot to still collect sufficient data to detect bottles and obstacles in its field of view without being bottlenecked by how fast inference takes.

Another requirement for our software system involves the range at which we want our robot to be able to classify items accurately. Since our use case requirements define the robot's task of intaking up to 3 bottles within a 1 meter radius, we want our robot's bottle detection algorithm to be able to accurately identify and label objects from a maximum distance of 2 meters away.

As per our use case requirement of picking up 3 bottles within 1 meter, we assert a hardware requirement that RecycleBot should hold 3 bottles in its on-unit storage. Since empty plastic water bottles weigh around 20 grams each, the additional weight when carrying 3 bottles in its storage is largely negligible. The main factor limiting the amount of bottles RecycleBot can collect is its storage capacity, an area around 25 square centimeters. This holds approximately 3 bottes when accounting for differing shapes and crush levels.

We additionally have a hardware requirement of the robot being able to complete 10 runs of the 1 meter test consecutively. As a part of our testing structure, we derived averages of each task over 10 runs, and thus required our robot to be able to run the task 10 times in a row without any components losing power.

V. Design Trade Studies

A. *Software*

a. **YOLOv5**: YOLOv5 is a popular model architecture and algorithm for object detection [7]. We chose to fine tune a YOLOv5 model because it maintains accuracy and is computationally fast compared to other models.

b. **Drinking waste dataset:** To train our model for object detection, we researched many open source datasets and found the most suitable one [8]. This dataset contains various images of plastic bottles, aluminum cans, glass bottles, and milk bottles. Since it has images of plastic bottles of different shapes, sizes, and orientations and other drinking waste to be classified into as obstacles, we conclude that this data is suitable for our use case. It has annotations indicating the bounding box around each object and its label, which is compatible with YOLOv5. This saves us the work of making our own dataset from scratch.

c. **KCF object tracking algorithm**: We chose to include an object tracking algorithm in our software pipeline in order to increase the number of FPS processed for the robot to properly move toward the target and avoid obstacles. Rather than spending valuable seconds performing inference on every frame while the robot is moving towards the target, the object tracking would be able to trace the positions of the objects found during inference. We chose this particular tracking algorithm out of the 8 total tracking algorithms housed in the OpenCV library due tradeoffs between speed and accuracy [9]. In our original implementation, we used CSRT. The CSRT Tracker, which is a Discriminative Correlation Filter (with Channel and Spatial Reliability) based object tracker, is aided by spatial reliability maps that manipulate and select the filter with the highest quality for application to the tracked object's region, which increases the search area and includes robustness to track non-rectangular objects, however at the expense of a slower throughput [10]. However, we found that when using CSRT in our pipeline, we were not getting good distance readings from the LiDAR. Since the CSRT algorithm would take so long to return a reading on any frame, it slowed down the entire processing speed and caused us to sample frames very infrequently such that our distance readings were far too sparse and inconsistent. This made us decide to transition to a quicker tracking algorithm at the cost of accuracy. We chose the still quite accurate KCF (Kernelized Corelation Filters) Tracker, which utilizes overlap between a set of images from a bag to track and predict the movement of an object, and displays high speed and accuracy but an inability to continue tracking after the loss of the object [10]. Even though KCF performs poorly when attempting to track the target even when other objects overlap, and one of our original concerns was creating a system that would not lose track of the target plastic bottle as it moved towards it, we found that there were few enough instances of an object getting in the way of the plastic bottle target while the bot was moving that using KCF was the most beneficial on the overall speed and accuracy of the pipeline.

d. **Rotation**: We decided to have RecycleBot rotate about 36 degrees clockwise at the end of its detection cycle, when it fails to identify a plastic bottle within its field of vision or detects an obstacle. The motivation behind this decision comes from the range of vision of the RealSense L515, which has a depth field of view of 70 degrees horizontal by 55 degrees vertical (+/- 3 degrees). Rotating 36 degrees 10 times in order to search in a new field of vision not only allows for a discrete number of turns to cover the full 360 degrees around the robot, but also introduces redundancy at the peripherals of the L515 camera's field of vision in case the bottle detection algorithm fails to identify a bottle at the edge of vision.

B. *Hardware*

a. **LiDAR camera L515**: We chose to use a LiDAR camera to augment our object detection algorithm, in order to implement a more effective navigation system. Without the depth points generated by the LiDAR camera, we would have to manually calculate the distance that RecycleBot would need to travel based purely on the robot speed and the relative change in size of the detected object, which would introduce margins of error due to noise and tracking loss. Since the only RealSense camera available from ECE lending was the L515, and purchasing a D400 series RealSense at $200+ price points was out of our budget given the sum of the materials that would needed to construct the intake and support on the iRobot, we decided to go with the L515 as our vision system. Since we test indoors for practicality, we concluded that the L515 would be sufficient. The L515 provides a horizontal depth field of vision of 70 degrees, which is large enough to perceive a significant area within the 1 meter radius, as well as a depth frame rate of 30 FPS, which is more than enough for our software pipeline, as inference on a single frame takes around 1 second. The minimum distance to receive depth readings is around 25 cm, and with the projected distance between the camera mount and the intake mechanism being greater than that distance, the camera will be able to provide depth readings for the bottles even as they are swept up into the storage area. It also records at 2 MP, which provides sufficient color resolution to utilize an OpenCV tracking algorithm effectively. While a LiDAR camera is more suited for an indoor environment due to its laser scanning technology, we've designed our software using the Realsense SDK, which is compatible with all camera series in the RealSense line, including other cameras that showcase better performance in outdoor environments [11]. Therefore, the project can be modified when scaled to use a more appropriate camera. Despite the performance metrics listed above, during our testing we found the camera to be quite finicky and often highly inaccurate with our pipeline. For example, many of the depth readings would just always be 0.0 meters, or when we sampled the same object's movement while tracking a bottle, the distance readings would just drop off to 0.0 meters, with the last reading being at an average of 0.5 meters away from the bottle, which is both an accuracy and sampling speed issue. Both of these causes produced inaccurate results and often messed up our pipeline since we base the decision of the closest bottle as well as the timing of turning the intake on and moving towards the bottle all on the distance readings. In order to design around this roadblock, we shifted last minute from using absolute distances to using relative distances to determine which bottle was closer (bounding boxes higher in y axis of the field of vision would be perceived as further away) and banking on when the tracking algorithm lost track of a bounding box to time turning on the intake.

b. **Jetson Xavier NX**: In our conversations deciding what we wanted as the brains of the system, we considered the Nvidia Jetson family and RaspberryPi's. However, we quickly discovered that for running multiple specialized tasks such as our computer vision and machine learning requirements, the unit most capable of, suited to, and documented for the computing demands we needed was a Jetson. With the knowledge that NVIDIA Jetsons are used for computer vision and neural network applications including image classification and object detection, we initially planned on utilizing the Jetson Nano, which has a 4-core CPU and 128-core Maxwell GPU with 4GB of memory, and runs on 5 Watts [12]. Its small size and low power consumption were two of our initial motivations for choosing it as our computing element. However, after considering other options in the Jetson line, combined with the discovery of the availability of a Jetson Xavier NX from ECE lending, we switched to using a Jetson Xavier NX. With a 6-core CPU and 384-core NVIDIA GPU with 8GB of memory, running on 10W, the Jetson Xavier NX holds far more processing power but is still housed on a unit the same size as the Nano [13]. Given its computing power advantage over the Nano, we concluded that the Xavier NX would be the best choice for our computer, given its size, low power consumption, and more extensive CPU and GPU. These features, combined with its pre-installed OpenCV, made it the most logical choice for our use case.

c. **iRobot**: We pivoted from planning to build the entire robot from a Rev Robotics kit to working with an iRobot Create 2 instead to cut down on the amount of time spent on building robot chassis that would take away from our time allocation for developing the intake mechanism. This tradeoff not only reduced build time but also effort required to implement a motor control system—iRobot has a very well documented Open Interface for control through UART [14], and an easier to work with Python wrapper library [18]. One issue we ran into was the minimum rotation degree of the iRobot. We initially experimented with sending rotation commands to the iRobot at lower and lower motor speeds where the left and right motors would be negatives of each other but found that after decreasing below around 15, the iRobot would not start movement at all, but any higher than 15 the iRobot would make a large sweeping rotation that would cause severe overcorrection issues when trying to center the bot in front of the bottle. We found that adding a slight bit of forward movement by offsetting one motor's speed by increments of 1 or 2 to the other was enough to provide the iRobot enough momentum to complete the minimal rotation. This discovery was a large breakthrough towards the end of our development process as it solved an issue we had been struggling with for quite a long time.

d. **Rotating intake**: We spent several weeks working through the design of the rotating intake, iterating through several implementations before settling on a shaft with rotating rubber wheels. One suggested method was a gate collection method, where a rising and falling gate would close to capture bottles that roll into the ground level intake opening. However, since this implementation would cause issues with the bottles potentially falling out of the storage, we went back to a previous idea with a rotating intake because it requires less additional structure and the intake itself blocks the collected bottles from rolling out. Through testing we have determined that the rotating intake requires sufficient speed to spin the bottles in the correct direction, rather than bumping it away from the intake. The actual material used to build the intake was decided based on what was readily available on popular robotics parts websites (AndyMark, Rev Robotics) and if the part sizes worked together well with the rest of the system. Many complaint wheels with different durometers were available, but we chose the ones that are the most rubber-like and squishy. These qualities are better for an intake because the wheels will be able to conform around whatever is being intaked when the motor is spinning fast. There are also many shafts and wheel holes available and we chose a ½ inch hex because many of the other components were compatible with ½ inch hex such as the shaft connector to connect the motor and shaft.

e. **Robot Construction Materials**: The center and side beams could have been built from wood or a metal such as aluminum. Although aluminum is slightly lighter and usually comes with more premade mounting holes, we decided to go with wood. This is because wood provides more flexibility with design because it is easier to work with. Furthermore to cut aluminum we would need something like a CNC machine, which requires more training and overhead to use. Before physically prototyping and building the RecycleBot we didn't know exactly how long the side beams should be because we didn't know the correct angle of the ramp that would allow the water bottle to easily go up it. After building the intake and running it we were able to gauge the angle of the ramp in order to easily allow water bottles to go up it. Once the angle was figured out and the ramp was attached to the side beams we needed to trim them so that the ramp was close to the intake. This way the water bottle would go up the ramp as soon as it touches the intake.

f. **Battery placement and center of mass**: The RecycleBot was very front heavy before the batteries were added to the back. This is because of the extensive weight of the intake and motor in the front. Originally we were going to mount the batteries next to the hardware in the front on the top lexan plate, but realized this would skew the center of mass even more. Thus we decided to place the batteries on the back of the RecycleBot and build a storage space for them using extra wood and lexan.

VI. System Implementation

A. Subsystem A: ML model

a. **Training before runtime**: To get data in the format to train and validate our model, we parsed our Drinking Waste Classification dataset [8] so that plastic bottles are in the bottle class and all other items are in the obstacle class. Then we split 80% of images to be in training and 20% in validation, making sure to randomize and that both training and validation have an equal number of bottle and non-bottle images. We then did transfer learning on a pre-trained YOLOv5 model, freezing all the backbone layers. We chose to do transfer learning because of the relatively small size of our dataset, which is approximately 4000 images. After experimenting with different hyperparameters, we trained for 10 epochs on the YOLOv5s model, then did fine tuning for 10 epochs with the default hyperparameters in the hyp.VOC.yaml file in the YOLOv5 library. When given images from the Drinking Waste Classification dataset, The model classifies bottles with 99% accuracy and obstacles with 97% accuracy.

b. **Inference at runtime**: At runtime, we perform inference with tiling on sampled color frames from the Realsense. Tiling is the process of dividing an image into tiles and performing inference on each individual tile before stitching them back together for the original image. This allows the model to better detect smaller objects in high-resolution images. We use the SAHI: Slicing Aided Hyper Inference library [15] to split our original image into 9 tiles. Tiling boosts our model to classify bottles up to 2m away from the robot, which is our target distance. We filter out all classifications below .35 confidence to reduce the number of false positives.

B. Subsystem B: CV and LiDAR at runtime

a. **Depth Sensing with Intel RealSense LiDAR Depth Camera L515**: There are several uses for the LiDAR camera in our project. We use the color frames for inference and the depth readings to measure the distance between the robot and objects detected from inference. We anticipate taking an initial depth reading from RecycleBot's initial position to any bottles to locate the closest one, then continue to take readings as the bot moves towards the target in order to maintain an appropriate speed, and to turn on the intake a feasible distance away from the bottle in order to pick it up. The LiDAR camera also gives distance readings to avoid obstacles. We chose to use pyrealsense2, the Python wrapper library for the RealSense SDK [16], to interface with the LiDAR Camera due to our familiarity with Python based machine learning and computer vision applications. While pyrealsense2 was available to install through pip on JetPack 4.x when we hadn't flashed the Xavier NX with JetPack 5.0.2 yet, it had to be built from source on the JetPack 5.0.2 updated disk image.

b. **OpenCV Object Tracking**: The main purpose of using an OpenCV tracking algorithm (we chose KCF) is to track bottles and obstacles when the robot moves towards a target, as well as to assist with rotation when RecycleBot needs to turn to face a target water bottle. As a part of our navigation algorithm, once RecycleBot has selected a target water bottle to attempt to pick up, it will first rotate to face the target head on. This entails first having to determine which side the bottle is located in relative to the vertical center line within the bot's field of vision. Once the side is determined, commands are sent to the iRobot to begin to incrementally turn in that direction. OpenCV object tracking allows the robot to continue following the selected object as its bounding box changes within the robot's field of vision. Once the robot is aligned with the target, we command it to stop and perform the second inference. We use the bounding boxes from the second inference for OpenCV multi object tracking on all objects (target bottle and obstacle). This tracking combined with the liDAR allows the robot to see if there are any obstacles in the way or if the target bottle is within pickup distance as the robot moves forward.

C. *Subsystem C: Hardware and Robotics at runtime*

a. **Construction of Robot**: The iRobot is the base of the robot and then the rest is mounted on top or around it. To mount the other components to the iRobot we are using Wafer screws that are screwed from the bottom of the iRobot nameplate to the wood on top. The iRobot only has top mounting through the nameplate, so the rest of the structure will need to be supported by the wood beam in the center. The center wood beam is then connected to 2 side wood beams. These wood beams are leaning against the sides of the iRobot and the ground. Connected to the center wood beam and the side wood beams is a lexan sheet that goes on top of the iRobot. This will be the bottom of the water bottle storage area.

b. **Power**: The components of our system which require power are the Jetson Xavier NX, the intake mechanism DC gearmotor, the Arduino Uno, and the iRobot. Since the iRobot has a charging dock as well as an on-unit battery lasting 2 hours per charge, we did not need to provide extra power for it. Hence, we ordered a 11.1V 6000mAh power supply for the Jetson, which requires 12V and draws up to 15W of power, so on a single charge the power supply will sustain the Jetson for around 4 hours, which will be more than enough for our use cases. With the Jetson powered and the Arduino Uno connected to it through USB, we were able to avoid a separate power supply for the Arduino, as the Jetson powers it. The motor is controlled by the Arduino Uno through the Cytron MD30C motor driver, powered by a 18V Milwaukee Drill Battery that was stepped down to 12V using a step down converter.

c. **Intake Mechanism**: The intake mechanism is mounted in the front of the robot. The intake is connected to both wood beams using intake motor mounts. The intake mount plate holds our intake. There is a ½ inch hex shaft that holds the wheels for the intake. The wheels are green compliant wheels with a 35A durometer. Then spacers are used between the wheels so they are held in place. Then at the ends of the wheels there are ½ inch shaft collars to keep the wheels and spacers from moving. The entire shaft is connected to the motor plates using more shaft collars and bearings to allow the shaft to rotate. The shaft is directly connected to the motor, which is mounted on the left side of the robot. The motor is screwed into a motor mounting plate that is also screwed into the left side beam. A more detailed picture of the intake is seen below in Figure 4. During runtime of the robot, the Arduino Uno controlling the motor driver for the intake will only allow the intake to spin when RecycleBot is within a certain distance from the target bottle.

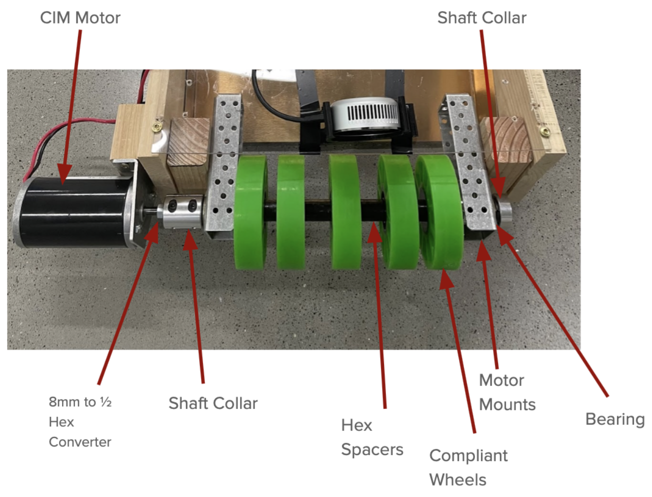


Fig. 4. Intake components

d. **Camera and Hardware Component Mounting**: All the electrical components and wiring are on top of a lexan plate that sits on top of the side wood beams. The camera is mounted in the middle of this lexan plate on our custom 3D printed stand. The other components mounted include the Nvidia Jetson Xavier NX, Arduino Uno, and motor driver. The wiring between all these components is also housed there.

e. **Water Bottle Storage Area**: In order to store more water bottles and prevent the water bottles from bouncing out of the robot we added 3 lexan plates on top of the iRobot. They are screwed into the side and middle wood beams. We then hot glued the 2 edges where the 3 lexan plates meet.

f. **Batteries and Step Down Converter**: There is a battery storage area on the back of the robot. This is made by extending the side beams a little using additional wood and a lexan backing screwed into the wood. This storage area is made so that the batteries are secured within the robot when it's moving and also easily removable to charge. The battery powering the Nvidia Jetson NX is velcroed to the back of the center wood beam and the drill battery powering the motor can be easily attached to the 18V to 12V step down converter. The main portion of the step down converter is screwed to the left side beam.

g. **iRobot Control**: As the iRobot moves near a bottle and it becomes within pickup range, it will activate the intake and then move forward towards the direction of the bottle to collect and push it up the ramp into the storage. If it is nearing an obstacle or if an obstacle is detected closer to the bot than a bottle is, RecycleBot will stop, turn 36 degrees clockwise, move forward .3m, turn 36 degrees counterclockwise, move forward again by .3m, and then restart the searching algorithm.

VII. Test, Verification and Validation

For testing, we have fixed some conditions in the environment:

1. **Objects are randomly placed within a 1 meter radius**. We expect our computer vision to detect items up to 2 meters away and a lack of GPS in our system constrains RecycleBot to only cover this amount of area. With a GPS, we can compose any grid of points 2 meters away from each other. For each point, we can use the same software to detect bottles and therefore, RecycleBot can be scalable with GPS.
2. **Objects are at least .45 meters apart from each other**. This condition allows the robot to have a chance to successfully avoid obstacles. If an obstacle and bottle are placed too close to each other, our robot intake will not be able to pick up a bottle while avoiding the obstacle. We also do not expect bottles and obstacles to be so closely clustered in the real world.
3. **RecycleBot will operate on a concrete background**. We want to minimize our scope to only concrete background in accordance with our use case requirements.
4. **RecycleBot will operate under fixed lighting conditions**. Our detection model training dataset was under fixed lighting conditions, so the performance with darker lighting is unknown.

Our test site was an empty area in Tech Spark, CMU's makerspace, which fit conditions 3 and 4 listed above. To evaluate our design, we have constructed two test cases, which only differ in the items placed within the 1 meter radius:

1. **3 bottles**
2. **3 bottles and 3 obstacles**

Figure 5 visualizes our two test cases.

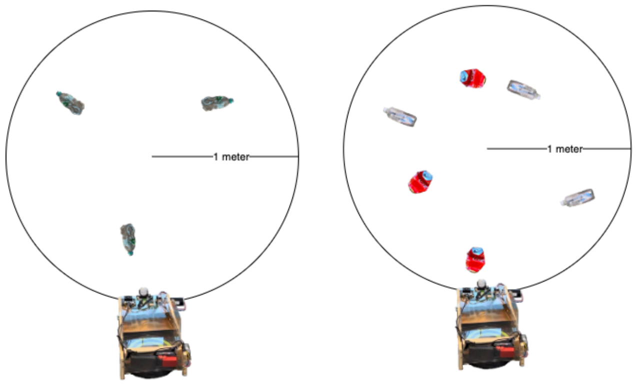


Fig. 5. Test environments 1 and 2

We define one run to be the completion of one 1 meter radius. For each run, we allow the robot to keep running until it reaches its stop state. Since we have a small sample size of 3 bottles per run, we will compare the average of 10 runs with

Table 1. Robot testing performance

Testing	Metrics (average of 10 runs)	Robot Performance
1. 3 bottles	90% bottle detection accuracy 70% pickup success <75 seconds for an average run <10% false positive rate for bottles	100% bottle detection accuracy 93.33% pickup success 89.41 seconds for an average run 0% false positive rate for bottles
2. 3 bottles and 3 obstacles	90% bottle detection accuracy 80% obstacle detection accuracy 80% obstacle avoidance success 70% pickup success	91.67% bottle detection accuracy 100% obstacle detection accuracy 50% obstacle avoidance success 100% pickup success

our metrics. The metrics and final performance of the robot are indicated in Table 1.

A. xBottle detection accuracy (1 & 2)

A bottle is considered detected if it is classified as a bottle at any time during inference. We achieved good bottle detection accuracy on both tests, 100% and 91.67% respectively. The bottle detection accuracy for Test 2 was lower than that of Test 1 because sometimes the obstacles blocked the robot's view of a bottle.

B. Pickup success (1 & 2)

An item is considered to be picked up if the intake turns on as the robot approaches the target and the target is swept underneath the intake. We only consider items that passed bottle detection in our pickup accuracies. Sometimes, the target would not make it all the way up the ramp or would fall out from the storage back down the ramp; however, we built the intake so that it blocks items from rolling out of the robot after they are inside, so these items would still be considered as a successful pickup. The intake's pickup success on the tests were 93.33% and 100% respectively.

C. False positive rate (1)

There were no false positives for bottles in Test 1. This is likely due to our test environment with a solid concrete background and fixed lighting so that there is a clear contrast between the bottle and the ground. We did notice 2 total obstacle false positives in Test 1, where a crack or dirty spots on the ground would be classified as an obstacle. This added to the speed of the system because the robot would move to avoid it but did not affect any other metrics.

D. Speed (1)

Speed was only evaluated in Test 1, as obstacles would introduce too much variance to the speed. The speed measured does not include setup time, such as initializing the object detection inference model, iRobot UART, and video stream from the Realsense. We achieved an average speed 14.41 seconds higher than our target speed metric due to additional pipeline logic that improved accuracy at the cost of speed. Originally, we did not anticipate needing a 2nd inference to detect potential obstacles that would be in view after angle calculation and we also planned to rotate only 6 times without seeing a bottle for the robot to enter its stopping condition. However, we realized during testing that rotating 6 times wasn't enough for the robot to see the entire area and opted to rotate 10 times in smaller increments.

E. Obstacle detection accuracy (2)

Our 3 obstacles consisted of a half gallon milk bottle, an orange juice bottle, and a Starbucks cup. We chose these obstacles because they are similar to those in the training dataset of our object detection model. An obstacle is considered detected if it is classified as an obstacle at any time during inference. We achieved 100% obstacle detection accuracy, likely due to our obstacle choices being consistent with the training dataset.

F. Obstacle avoidance success (2)

For obstacle avoidance, we only considered the cases where the obstacle was blocking the path of the robot. If the robot stops and redirects its path so that the obstacle is out of the way, it is a success. We did not pass this metric and only achieved a 50% obstacle avoidance success due to the inaccurate depth readings from the Realsense. If an obstacle was initially over 1 meter away from the robot, the depth readings would give 0 meter readings even when the robot approached closer to the obstacle. This was a major problem for obstacle avoidance because the robot would never be able to gauge the distance between itself and an obstacle that was more than 1 meter away from it.

VIII. Project Management

A. Schedule

See the Appendix on page 16 for a full Gantt chart.

B. Team Member Responsibilities

Table 2 shows the specific responsibilities of members of the team. Serena and Mae were responsible for integrating the software pipeline and everyone was responsible for testing.

Team Member	**Responsibilities**
Meghana	● Design and construct a structure on top of the iRobot ● Design and construct an intake ● Arduino to motor controls for the intake
Serena	● Jetson Xavier NX and RealSense setup and maintenance ● Depth perception development ● openCV object tracking and navigation development
Mae	● Find dataset for ML model ● Perform transfer learning to train a custom YOLOv5 model to detect bottles and obstacles ● Write logic for entire software system

Table 2. Team member responsibilities

C. Bill of Materials and Budget

See the Appendix on page 15 for the detailed bill of materials and budget. On a high level we built on top of iRobot Create 2. We used wood to build the structure around the iRobot and connect the iRobot to our intake. We are also using aluminum and lexan for the ramp and bottom of the storage area. On the hardware side, we are using an NVIDIA Jetson Xavier NX, Arduino Uno, a motor driver, and Intel Realsense. The other intake and construction materials have been purchased from Andy Mark, Amazon, Home Depot and GoBuilda.

D. Risk Management

One major risk was the physical construction of the structure around the iRobot. We needed to balance weight and stability, as the maximum weight the iRobot can hold is 15-20 lbs [17]. To mitigate the risk, we bought 2 different types of wood, a heavier and lighter wood so we could test whether the lighter wood was stable enough while having backup if it did not work. We picked out other building materials carefully to try to minimize weight and designed to balance weight throughout the robot. Since we were drilling or using velcro to attach components together, we could easily take them apart and rebuild parts of the robot if something wasn't working.

For the software, we had a couple major concerns. The object detection algorithm was not guaranteed to work on bottles at a far distance. We had several ideas to implement to get the robot to detect objects at least 2 meters away: generating more training data using images captured by the Realsense, increasing color image resolution, and slicing the image to perform inference on smaller sections of the original image. Since we found a library that does slicing and is compatible with YOLO, we tried that method first and it was a success. Another issue was the accuracy of the Realsense's liDAR depth frames. If the depth readings were not accurate, we would resort to the relative positioning of objects in the color frame to get relative distance between the robot and objects. We did have to do this in some parts of the pipeline.

An unforeseen problem we had was the setup of everything we needed in the Xavier NX. Halfway through the semester, after we finished our first iteration of the pipeline, we discovered that inference was taking around 20 seconds each frame, which was highly undesirable. We found that the reason why the Xavier NX was not performing as expected was that it was not actually utilizing GPU. This turned out to be an issue with not having CUDA availability due to the JetPack version on the Xavier NX not being updated to JetPack 5.0. While attempting to figure out the root cause of the Xavier NX's slow speed, we went through much trial and error, including: going back to the Jetson Nano, which after flashing with the JetPack 5.0 disk image, had too small of an SD card size to support any reasonable functionality from our pipeline; flashing the JetPack 5.0 disk image to the Xavier NX but running into Python3 issues while finding that the pyrealsense2 wrapper library for the RealSense L515 had to be built from source and could not be installed using pip3; trying to roll back to JetPack 4.x on the but discovering that in the process of updating to JetPack 5.0 we had overwritten on-board memory separate from the SD card and could not reflash the board unless we had access to the Jetson SDK manager which was not compatible with JetPack 4.x versions; and finally reflashing a fresh JetPack 5.0.2 disk image onto the Xavier NX and building the pyrealsense2 library from source, which additionally required extra research into installing dependencies such as CMake. Overall, this process of accessing CUDA on the Xavier NX was frustrating and time consuming for over two weeks but once correctly configured, cut down our inference time from around 20 seconds to around 1 second, a 40x speedup.

To ensure the on-time completion of our project, we split our test cases so we could easily rescope and not do obstacle avoidance, making the software pipeline much easier. We built in ample slack time to resolve problems that might come up or pivot to another implementation.

IX. Ethical Issues

Since the RecycleBot is meant to be used in public spaces commonly littered with plastic bottles, the main concern is that the spinning intake could be dangerous for people, especially children. Thus, to prevent harm the robot should be used when an area is closed and sectioned off to the public. This solution also prevents people from maliciously putting dangerous objects near the intake as it is spinning. Furthermore, the intake could also harm the robot if it attempts to pick up a sharp object, such as a broken glass bottle. More testing should be done to ensure that the object detection algorithm performs well. Since it may be impossible to have a perfect classification rate, monthly maintenance and inspection should occur to see whether parts are damaged and need to be replaced.

Another ethical concern is if someone were to hack into the system and steal video footage or control the robot for their own malicious purposes. Stealing data could violate the privacy of people caught on camera and taking control of the robot could have dangerous consequences. The robot's network should be secure and safe from third parties breaking in.

X. Related Work

A variety of trash and recycle sorting robots exist, one of which is TrashBot [3]. TrashBot is a trash bin that is able to distinguish between trash and recyclable items. After throwing an item into TrashBot, it uses metal and computer vision to determine whether or not an item is recyclable. Upon classification, it uses a system of trap doors to discard the item into the correct bin. Such mechanisms help in the autonomous classification of trash, but still require people to properly dispose of their waste and therefore, they do not solve the problem of reducing existing litter in public places.

Many autonomous cleaning robot solutions with computer vision exist in today's industry. ClearBot, a robot that detects and collects litter in local waterways, has an autonomous setting where it uses computer vision to locate and collect floating pieces of litter [4]. Another such autonomous cleaning robot is BeachBot, a land operated robot designed to pick up cigarette butts on beaches [5]. The bot uses artificial intelligence to distinguish cigarette butts in the sand, then picks them up with grippers, and into an internal storage. It is also designed to avoid obstacles. Such robots are very similar to RecycleBot but have two main differences. First, the locations of operation differ since RecycleBot is meant to operate on hard, concrete surfaces. Second, these robots collect trash whereas RecycleBot collects only recyclable items, reducing the need to sort items into trash and recycling after collection.

XI. Summary

We created an efficient robot that will autonomously detect and collect standard size plastic bottles from the ground. Users can simply leave the robot where they want and the robot will clean, reducing manpower in picking up litter. We have met all of our system requirements excluding speed and obstacle avoidance. To improve speed and maintain similar accuracy levels, some options are upgrading to a Jetson with higher compute power such as the Jetson Xavier AGX or reducing the time of the second inference in the software pipeline by using a smaller, faster model since the second inference is only used to catch any obstacles in view after angle calculation and does not need to be fine tuned to classify bottles. For obstacle avoidance, our main issue was the limitation of the Intel Realsense L515, as it was not able to capture accurate and reliable distances, especially when an object was off-center or more than 1 meter away. Obstacle avoidance could be improved with better depth readings and a path recalculation algorithm that would adjust for how large the obstacle is.

Through this project we have learned how to use what we have learned throughout our time at CMU to design and integrate components to create a complete solution. We learned not to underestimate software setup time, as we struggled to flash the Jetson and make sure the JetPack and Python versions were compatible to utilize our Jetson's GPU and install all the libraries we needed. Overall, this was a great learning experience and we are proud of our work.

Glossary of Acronyms

- **CSRT - Discriminative Correlation Filter with Channel and Spatial Reliability**
- **COCO - Common Objects in Context, a publicly available image dataset collected for the goal of advancing image recognition**
- **FPS - Frames per second**
- **LiDAR - Light Detection and Ranging**
- **UART - Universal Asynchronous Receiver-Transmitter**
- **WAAS - Wide Area Augmentation System**
- **Yolo(v5) - 'You Only Look Once' Version 5, a family of object detection models trained on the COCO dataset which we applied transfer learning to in order to fine tune our bottle detection model**

References

[1] George, Lowell. "The Five Most Common Things Found at River Cleanups." *American Rivers*, 2 Jan. 2020, https://www.americanrivers.org/2018/01/five-common-things-found-river-cleanups/.

[2] *Plastic water bottle pollution: Where do all the bottles end up?* (no date) *Plastic Water Bottle Pollution: Where Do All The Bottles End Up? – Healthy Human*. Available at: https://healthyhumanlife.com/blogs/news/plastic-water-bottle-pollution-plastic-bottles-end (Accessed: December 17, 2022).

[3] *Trashbot: The smart recycling bin that sorts at the point of disposal* (2022) *CleanRobotics*. Available at: https://cleanrobotics.com/trashbot/ (Accessed: December 17, 2022).

[4] Andrea.d.steffen (2021) *This autonomous zero-emissions robot collects garbage from waterways*, *Intelligent Living*. Available at: https://www.intelligentliving.co/autonomous-robot-collects-garbage-from-waterways/ (Accessed: December 17, 2022).

[5] *Meet BeachBot, a beach rover that uses AI to remove cigarette butts from beaches* (2021) *Business Insider*. Available at: https://www.businessinsider.in/tech/news/meet-beachbot-a-beach-rover-that-uses-ai-to-remove-cigarette-butts-from-beaches/articleshow/84759758.cms (Accessed: December 17, 2022).

[6] Sensor types (modalities) for robots to experience the world | Robots For Roboticists says: *et al.* (2016) *GPS and global absolute positioning*, *Robots For Roboticists*. Available at:
https://www.robotsforroboticists.com/gps/#:~:text=Wide%20Area%20Augmentation%20System%20(WAAS)&text=In%20practice%20WAAS%20assisted%20GPS,the%20sky%20you%20can%20see (Accessed: December 17, 2022).

[7] Ultralytics (no date) *Ultralytics/yolov5: Yolov5 🚀 in PyTorch > ONNX > CoreML > TFLite*, *GitHub*. Available at: https://github.com/ultralytics/yolov5 (Accessed: December 17, 2022).

[8] Serezhkin, A. (2020) *Drinking waste classification*,

Kaggle. Available at: https://www.kaggle.com/datasets/arkadiyhacks/drinking-waste-classification (Accessed: December 17, 2022).

[9] Rosebrock, A. (2022) *OpenCV object tracking*, *PyImageSearch*. Available at: https://pyimagesearch.com/2018/07/30/opencv-object-tracking (Accessed: December 17, 2022).

[10] BroutonLab (2020) *A complete review of the opencv object tracking algorithms*, *BroutonLab*. BroutonLab. Available at: https://broutonlab.com/blog/opencv-object-tracking (Accessed: December 17, 2022).

[11] *Intel® realsense™ Lidar Camera L515* (2022) *Intel® RealSense™ Depth and Tracking Cameras*. Available at: https://www.intelrealsense.com/lidar-camera-l515/ (Accessed: December 17, 2022).

[12] *Jetson Nano Developer Kit* (2022) *NVIDIA Developer*. Available at: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (Accessed: December 17, 2022).

[13] *The World's smallest AI supercomputer* (no date) *NVIDIA*. Available at: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ (Accessed: December 17, 2022).

[14] *IRobot Roomba 600 open interface spec - irobotweb.com* (no date). Available at: https://www.irobotweb.com/%20~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf?%20la%20=%20es (Accessed: December 17, 2022).

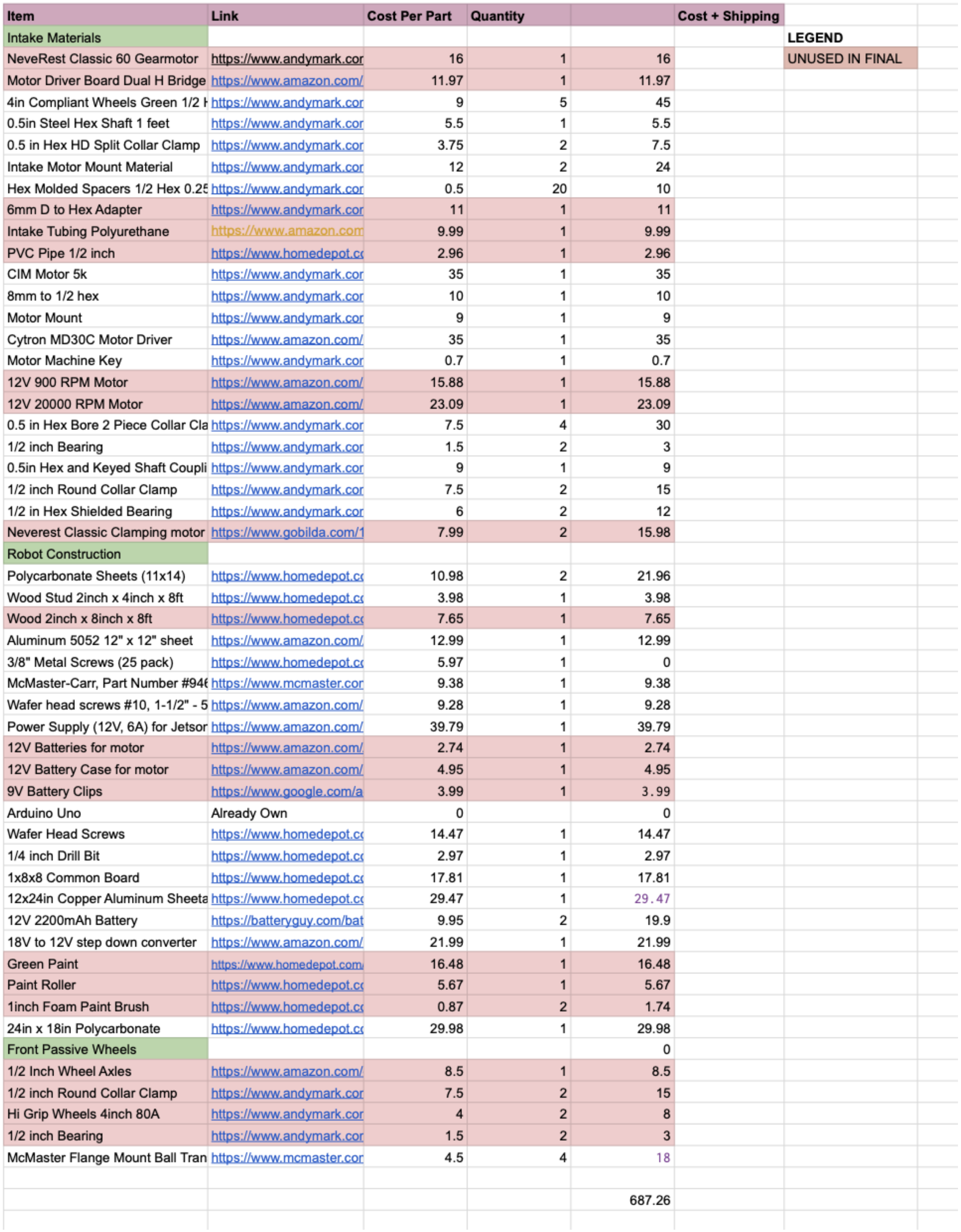
[15] Obss (no date) *OBSS/Sahi: Framework agnostic sliced/tiled inference + interactive UI + error analysis plots*, *GitHub*. Available at: https://github.com/obss/sahi (Accessed: December 17, 2022).

[16] *Pyrealsense2* (no date) *PyPI*. Available at: https://pypi.org/project/pyrealsense2/ (Accessed: December 17, 2022).

[17] *How much weight can the irobot carry?*, *Robotics Stack Exchange*. Available at: https://robotics.stackexchange.com/questions/10686/how-much-weight-can-the-irobot-carry (Accessed: December 17, 2022).

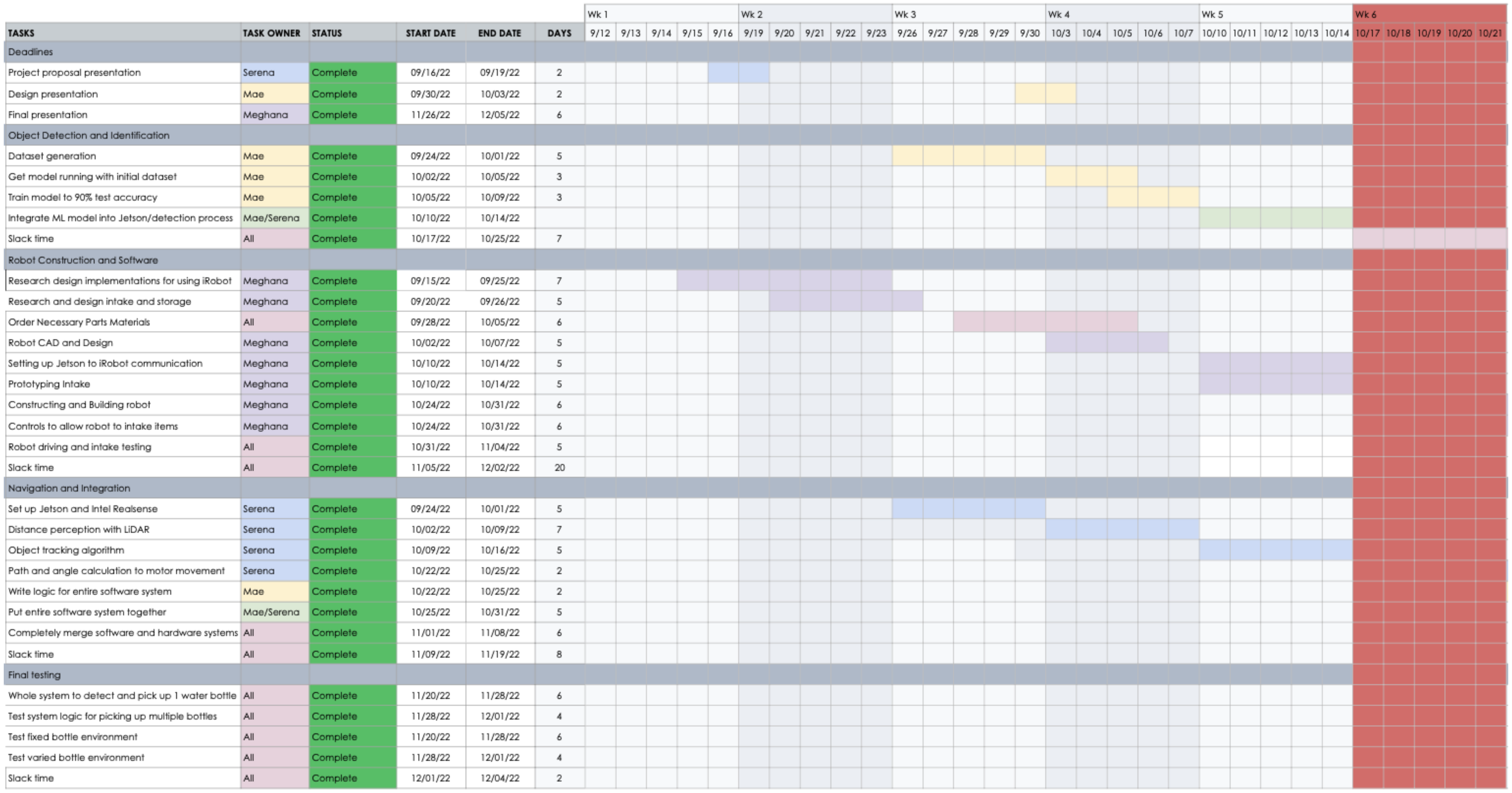
[18] *GitHub - momsfriendlyrobotcompany/pycreate2: Library for irobot create 2* (no date). Available at: https://github.com/MomsFriendlyRobotCompany/pycreate2 (Accessed: December 17, 2022).

APPENDIX- Bill of materials

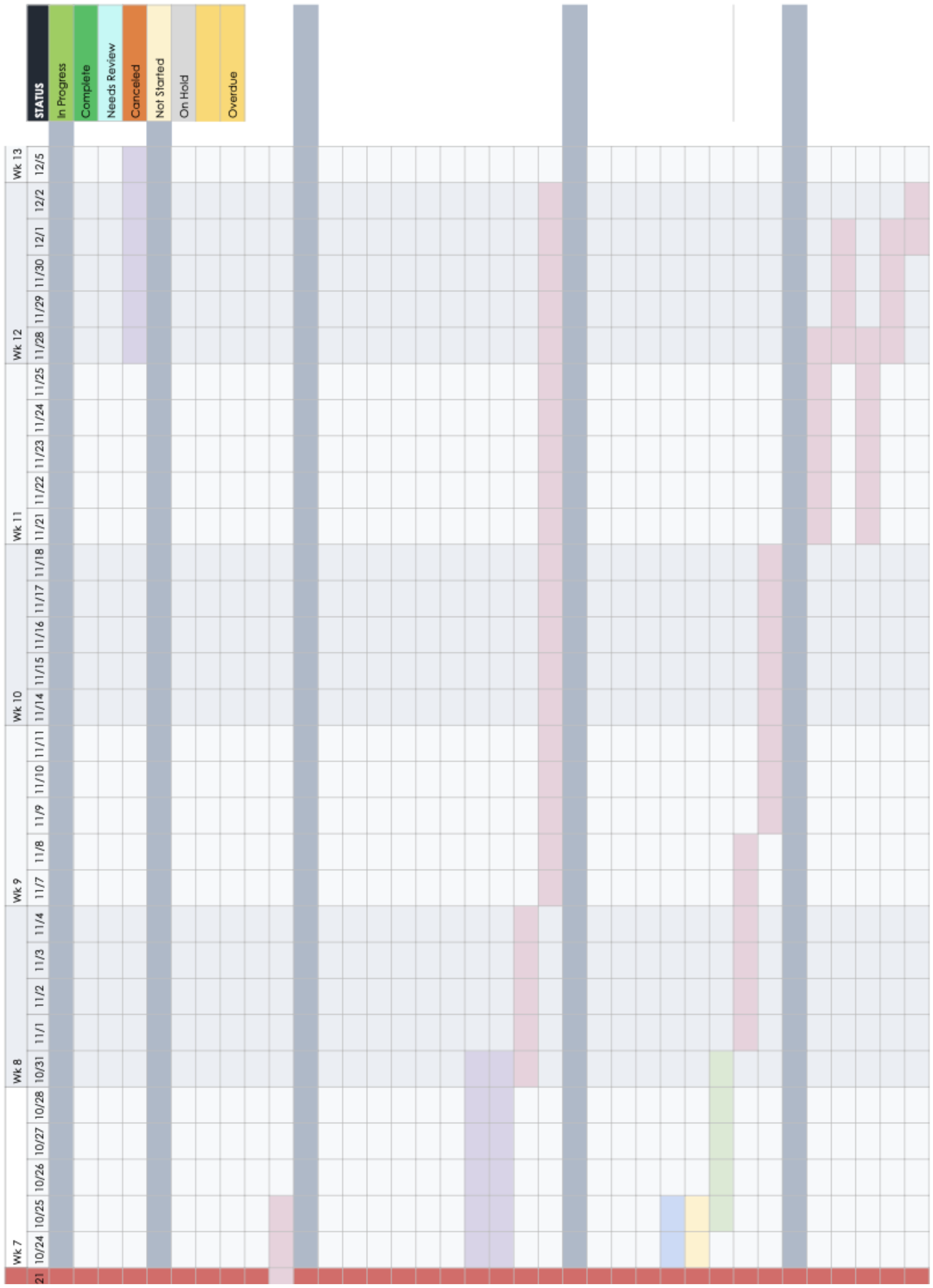


Item	Link	Cost Per Part	Quantity		Cost + Shipping	
Intake Materials						LEGEND
NeveRest Classic 60 Gearmotor	https://www.andymark.cor	16	1	16		🟥 UNUSED IN FINAL
Motor Driver Board Dual H Bridge	https://www.amazon.com/	11.97	1	11.97		
4in Compliant Wheels Green 1/2 H	https://www.andymark.cor	9	5	45		
0.5in Steel Hex Shaft 1 feet	https://www.andymark.cor	5.5	1	5.5		
0.5 in Hex HD Split Collar Clamp	https://www.andymark.cor	3.75	2	7.5		
Intake Motor Mount Material	https://www.andymark.cor	12	2	24		
Hex Molded Spacers 1/2 Hex 0.25	https://www.andymark.cor	0.5	20	10		
6mm D to Hex Adapter	https://www.andymark.cor	11	1	11		
Intake Tubing Polyurethane	https://www.amazon.com	9.99	1	9.99		
PVC Pipe 1/2 inch	https://www.homedepot.cc	2.96	1	2.96		
CIM Motor 5k	https://www.andymark.cor	35	1	35		
8mm to 1/2 hex	https://www.andymark.cor	10	1	10		
Motor Mount	https://www.andymark.cor	9	1	9		
Cytron MD30C Motor Driver	https://www.amazon.com/	35	1	35		
Motor Machine Key	https://www.andymark.cor	0.7	1	0.7		
12V 900 RPM Motor	https://www.amazon.com/	15.88	1	15.88		
12V 20000 RPM Motor	https://www.amazon.com/	23.09	1	23.09		
0.5 in Hex Bore 2 Piece Collar Cla	https://www.andymark.cor	7.5	4	30		
1/2 inch Bearing	https://www.andymark.cor	1.5	2	3		
0.5in Hex and Keyed Shaft Coupli	https://www.andymark.cor	9	1	9		
1/2 inch Round Collar Clamp	https://www.andymark.cor	7.5	2	15		
1/2 in Hex Shielded Bearing	https://www.andymark.cor	6	2	12		
Neverest Classic Clamping motor	https://www.gobilda.com/1	7.99	2	15.98		
Robot Construction						
Polycarbonate Sheets (11x14)	https://www.homedepot.cc	10.98	2	21.96		
Wood Stud 2inch x 4inch x 8ft	https://www.homedepot.cc	3.98	1	3.98		
Wood 2inch x 8inch x 8ft	https://www.homedepot.cc	7.65	1	7.65		
Aluminum 5052 12" x 12" sheet	https://www.amazon.com/	12.99	1	12.99		
3/8" Metal Screws (25 pack)	https://www.homedepot.cc	5.97	1	0		
McMaster-Carr, Part Number #946	https://www.mcmaster.cor	9.38	1	9.38		
Wafer head screws #10, 1-1/2" - 5	https://www.amazon.com/	9.28	1	9.28		
Power Supply (12V, 6A) for Jetsor	https://www.amazon.com/	39.79	1	39.79		
12V Batteries for motor	https://www.amazon.com/	2.74	1	2.74		
12V Battery Case for motor	https://www.amazon.com/	4.95	1	4.95		
9V Battery Clips	https://www.google.com/a	3.99	1	3.99		
Arduino Uno	Already Own	0		0		
Wafer Head Screws	https://www.homedepot.cc	14.47	1	14.47		
1/4 inch Drill Bit	https://www.homedepot.cc	2.97	1	2.97		
1x8x8 Common Board	https://www.homedepot.cc	17.81	1	17.81		
12x24in Copper Aluminum Sheeta	https://www.homedepot.cc	29.47	1	29.47		
12V 2200mAh Battery	https://batteryguy.com/bat	9.95	2	19.9		
18V to 12V step down converter	https://www.amazon.com/	21.99	1	21.99		
Green Paint	https://www.homedepot.com/	16.48	1	16.48		
Paint Roller	https://www.homedepot.cc	5.67	1	5.67		
1inch Foam Paint Brush	https://www.homedepot.cc	0.87	2	1.74		
24in x 18in Polycarbonate	https://www.homedepot.cc	29.98	1	29.98		
Front Passive Wheels				0		
1/2 Inch Wheel Axles	https://www.amazon.com/	8.5	1	8.5		
1/2 inch Round Collar Clamp	https://www.andymark.cor	7.5	2	15		
Hi Grip Wheels 4inch 80A	https://www.andymark.cor	4	2	8		
1/2 inch Bearing	https://www.andymark.cor	1.5	2	3		
McMaster Flange Mount Ball Tran	https://www.mcmaster.cor	4.5	4	18		
				687.26		

APPENDIX-Schedule



						Wk 1					Wk 2					Wk 3					Wk 4					Wk 5					Wk 6				
TASKS	TASK OWNER	STATUS	START DATE	END DATE	DAYS	9/12	9/13	9/14	9/15	9/16	9/19	9/20	9/21	9/22	9/23	9/26	9/27	9/28	9/29	9/30	10/3	10/4	10/5	10/6	10/7	10/10	10/11	10/12	10/13	10/14	10/17	10/18	10/19	10/20	10/21
Deadlines																															🟥	🟥	🟥	🟥	🟥
Project proposal presentation	Serena	Complete	09/16/22	09/19/22	2					■	■																				🟥	🟥	🟥	🟥	🟥
Design presentation	Mae	Complete	09/30/22	10/03/22	2															■	■										🟥	🟥	🟥	🟥	🟥
Final presentation	Meghana	Complete	11/26/22	12/05/22	6																										🟥	🟥	🟥	🟥	🟥
Object Detection and Identification																															🟥	🟥	🟥	🟥	🟥
Dataset generation	Mae	Complete	09/24/22	10/01/22	5											■	■	■	■	■											🟥	🟥	🟥	🟥	🟥
Get model running with initial dataset	Mae	Complete	10/02/22	10/05/22	3																■	■	■								🟥	🟥	🟥	🟥	🟥
Train model to 90% test accuracy	Mae	Complete	10/05/22	10/09/22	3																		■	■	■						🟥	🟥	🟥	🟥	🟥
Integrate ML model into Jetson/detection process	Mae/Serena	Complete	10/10/22	10/14/22																						■	■	■	■	■	🟥	🟥	🟥	🟥	🟥
Slack time	All	Complete	10/17/22	10/25/22	7																										■	■	■	■	■
Robot Construction and Software																															🟥	🟥	🟥	🟥	🟥
Research design implementations for using iRobot	Meghana	Complete	09/15/22	09/25/22	7				■	■	■	■	■	■	■																🟥	🟥	🟥	🟥	🟥
Research and design intake and storage	Meghana	Complete	09/20/22	09/26/22	5							■	■	■	■	■															🟥	🟥	🟥	🟥	🟥
Order Necessary Parts Materials	All	Complete	09/28/22	10/05/22	6													■	■	■	■	■	■								🟥	🟥	🟥	🟥	🟥
Robot CAD and Design	Meghana	Complete	10/02/22	10/07/22	5																■	■	■	■							🟥	🟥	🟥	🟥	🟥
Setting up Jetson to iRobot communication	Meghana	Complete	10/10/22	10/14/22	5																					■	■	■	■	■	🟥	🟥	🟥	🟥	🟥
Prototyping Intake	Meghana	Complete	10/10/22	10/14/22	5																					■	■	■	■	■	🟥	🟥	🟥	🟥	🟥
Constructing and Building robot	Meghana	Complete	10/24/22	10/31/22	6																										🟥	🟥	🟥	🟥	🟥
Controls to allow robot to intake items	Meghana	Complete	10/24/22	10/31/22	6																										🟥	🟥	🟥	🟥	🟥
Robot driving and intake testing	All	Complete	10/31/22	11/04/22	5																										🟥	🟥	🟥	🟥	🟥
Slack time	All	Complete	11/05/22	12/02/22	20																										🟥	🟥	🟥	🟥	🟥
Navigation and Integration																															🟥	🟥	🟥	🟥	🟥
Set up Jetson and Intel Realsense	Serena	Complete	09/24/22	10/01/22	5											■	■	■	■	■											🟥	🟥	🟥	🟥	🟥
Distance perception with LiDAR	Serena	Complete	10/02/22	10/09/22	7																■	■	■	■	■						🟥	🟥	🟥	🟥	🟥
Object tracking algorithm	Serena	Complete	10/09/22	10/16/22	5																					■	■	■	■	■	🟥	🟥	🟥	🟥	🟥
Path and angle calculation to motor movement	Serena	Complete	10/22/22	10/25/22	2																										🟥	🟥	🟥	🟥	🟥
Write logic for entire software system	Mae	Complete	10/22/22	10/25/22	2																										🟥	🟥	🟥	🟥	🟥
Put entire software system together	Mae/Serena	Complete	10/25/22	10/31/22	5																										🟥	🟥	🟥	🟥	🟥
Completely merge software and hardware systems	All	Complete	11/01/22	11/08/22	6																										🟥	🟥	🟥	🟥	🟥
Slack time	All	Complete	11/09/22	11/19/22	8																										🟥	🟥	🟥	🟥	🟥
Final testing																															🟥	🟥	🟥	🟥	🟥
Whole system to detect and pick up 1 water bottle	All	Complete	11/20/22	11/28/22	6																										🟥	🟥	🟥	🟥	🟥
Test system logic for picking up multiple bottles	All	Complete	11/28/22	12/01/22	4																										🟥	🟥	🟥	🟥	🟥
Test fixed bottle environment	All	Complete	11/20/22	11/28/22	6																										🟥	🟥	🟥	🟥	🟥
Test varied bottle environment	All	Complete	11/28/22	12/01/22	4																										🟥	🟥	🟥	🟥	🟥
Slack time	All	Complete	12/01/22	12/04/22	2																										🟥	🟥	🟥	🟥	🟥


STATUS
In Progress
Complete
Needs Review
Canceled
Not Started
On Hold
Overdue
Wk 7
Wk 8
Wk 9
Wk 10
Wk 11
Wk 12
Wk 13
21
10/24
10/25
10/26
10/27
10/28
10/31
11/1
11/2
11/3
11/4
11/7
11/8
11/9
11/10
11/11
11/14
11/15
11/16
11/17
11/18
11/21
11/22
11/23
11/24
11/25
11/28
11/29
11/30
12/1
12/2
12/5