RecycleBot, an Automatic Smart Trash Sorting Can

Brian Liao*, Haoyan Huo†, Yunfeng Gao‡ and Alex Kiani§
Department of Electrical Engineering and Computer Sciences, University of California Berkeley
Berkeley, CA, USA
Email: *btl787@berkeley.edu, †haoyan_huo@berkeley.edu, ‡yunfengg@berkeley.edu, §akiani@berkeley.edu

***Abstract*—RecycleBot is an automatic smart trash sorting can. A user can place their trash on a stand and RecycleBot will use Computer Vision to sort the trash into three categories: *Trash*, *Recycling*, or *Compost*. This is done by a Kobuki holding three trash bins, which rotates to align the correct bin to the opening on a enclosing trash box. RecycleBot can also deliver trash to a pickup location when it detects a bin is full. The delivery is performed by path finding on a colored tape fixed on the ground. RecycleBot is therefore able to make trash, recycling, and compost disposal more effective and efficient.**

***Index Terms*—Embedded Systems, Recycling, Compost, Trash, Self-Sorting, Computer Vision, Robotics, Sensors and Actuators, Networking, Continuous Dynamics, Finite-State Machines**

I. Objective

Our objective in designing RecycleBot was to create a smart trash bin that helps user recycle and dispose trash more effectively. Most times, people throw away recyclable material such as coffee lids because they do not know it is recyclable. Furthermore, if they throw non-recyclable material like styrofoam into the recycling bin, workers are required to go through the trash and separate out the non-recyclable trash.

Fig. 1. Example of trash bins for trash, recycling, and compost.

These problems can be solved by creating a smart trash bin that makes trash sorting seamless and effortless for users. Using RecycleBot, a user can place his trash on a stand and RecycleBot will take a picture with it's camera. It then uses computer vision (CV) to classify the trash for the user into trash, compost, or recycling. RecycleBot then aligns the correct trash bin for the user to throw away their trash.

In addition, RecycleBot can automatically take out trash. When RecycleBot detects one bin is full, it can perform path finding and drive itself to a specified location for a trash disposal man to pick up the trash.

We would like to express our thanks to the CS 149 GSIs Neal Jackson, Eddie Kim, and Jean-Luc Watson.

Building on top of our RecycleBot prototype, future engineers can use it to create industrial-grade smart trash cans that can be sold in a profitable startup. Features that can be polished include the CV system, material manufacturing and product construction, and self-takeout of trash bins.

For computer vision, we used a Google Cloud Vision (GCV) API that provides image labels of uploaded images. GCV attempts to label all general objects, which makes trash classification less accurate. For example, takeout containers may be made out of paper, but are *not* recyclable as they contain food grease that contaminates recycling. We could improve this by training our own fine grain model that will be more accurate at classifying trash.

Second, for our trash bin box, we had to make compromises on the material for the box. We chose to make the box using cardboard both due to budget constraints and difficulty of cutting wood, the original planned material for our trash box. A professional engineer could create a box that is suitable for mass-scale manufacturing using materials such as aluminium. The design of the appearance can also be improved to integrate the box's microprocessors and cameras within the box to make the product more professional.

Finally, RecycleBot performs automatic trash disposal by path finding to a specified location. In a production system, this can be extended to have multiple RecycleBots that all have paths to specified pick-up locations. This way, in a setting such as a sport stadium, trash can be picked up and disposed.

II. System Design

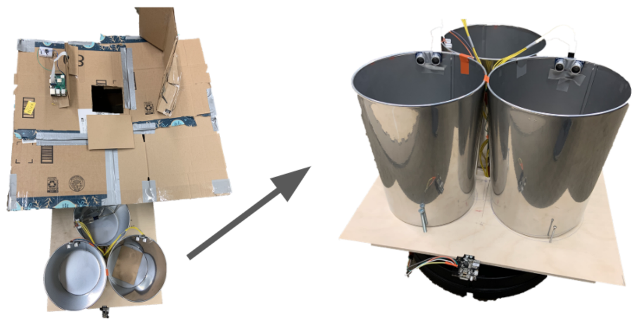


Fig. 2. RecycleBot trash box and Kobuki trash carrier system.

The final RecycleBot consists of two parts: a large box with a opening on the top for trash to fall into, and three trash bins installed on a Kobuki robot (will be named as *trash carrier* throughout this document), as shown in Fig. 2. The system architecture of RecycleBot is demonstrated in Fig. 3. The hardware electronics is divided into two microcontroller boards:

1) A Raspberry Pi board (any version), responsible for capturing raw camera pictures, classifying trash in pictures by executing GCV API, and sending the correct rotation commands to the Buckler board via Bluetooth Low Energy (BLE).
2) A Berkeley Buckler board mounted on a Yujin Kobuki robot, responsible for carrying three trash bins and act according to its environment and the rotation commands from the Raspberry Pi board.

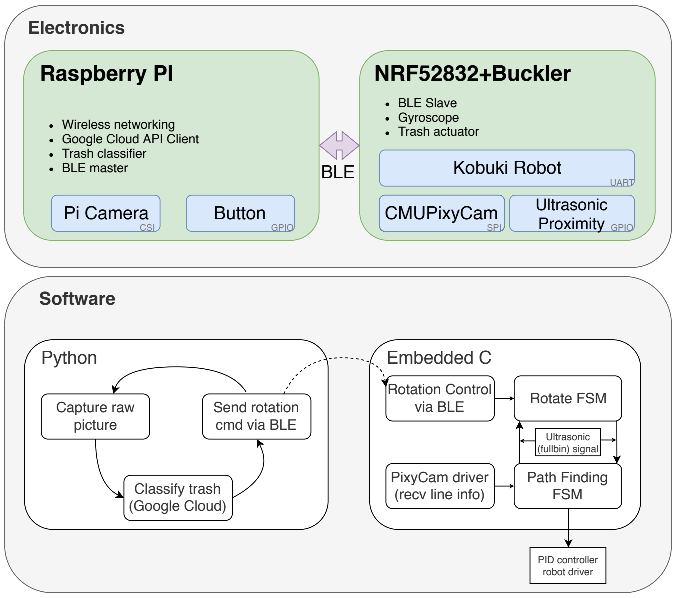


Fig. 3. Hardware and software architecture overview of RecycleBot.

As shown in the software diagram in Fig. 3, the software of this project is divided into several modules. For the Raspberry Pi board, we implemented all three modules in pure Python, running on the Linux-like OS, Raspbian. For the trash carrier, all modules are written in C, utilizing the open-source Buckler code and nRF5 libraries provided in the course labs.

A. Image Capture Module using Raspberry Pi Camera

We use the Raspberry Pi Camera Module V2 to take pictures of trash. The camera resolution is 1920x1080, a high enough resolution to accurately classify objects. We used the Python PiCamera library [3] to control the camera. The image is taken once a user has placed trash on a stand and pressed a switch to activate the camera. The status of the switch is detected by an interrupt handler for the GPIO port connected to the switch.

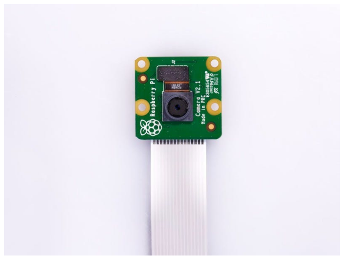


Fig. 4. Raspberry Pi Camera Module V2 used to capture trash pictures.

B. Computer Vision Classification Module

Once the image is taken, we call the GCV Image Classification API [2]. This API returns a list of labels it believes is in the image. For example, if an image of an apple is uploaded, the GCV API may return, “apple, fruit, red, sphere”. For each of our classes, trash, compost, recycling, and no object, we create a list of labels that would fit the class. Some examples for compost are, “fruit, fish, dairy, paper towels, coffee grounds.” To classify, we count how many labels returned by the GCV API fit in each class and classify the image as the one with the most labels for that class.

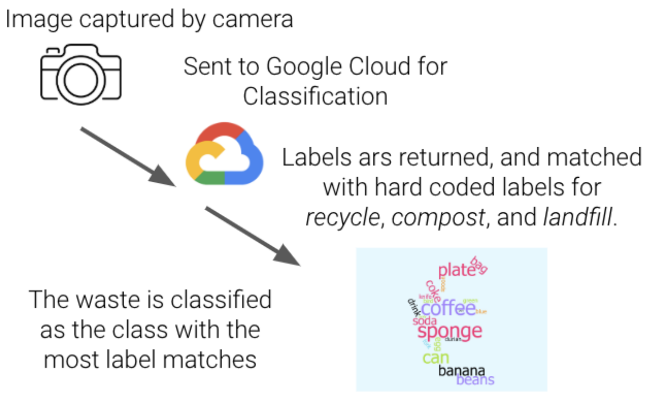


Image captured by camera
Sent to Google Cloud for Classification
Labels ars returned, and matched with hard coded labels for *recycle*, *compost*, and *landfill*.
The waste is classified as the class with the most label matches
plate
coffee
sponge
can
banana
beans

Fig. 5. Diagram of trash classification using Google Cloud.

C. BLE Communication Module

The trash carrier must align one of the three bins with the opening on the box once trash is to be thrown into. This action is performed by transmitting commands via BLE and executing it by the trash carrier. We implemented a BLE device with two characteristics: *rotation*, a floating number representing the desired rotation angle from a reference point, and *is_available*, a boolean value representing whether the trash carrier is available (residing inside the trash box).

Once the trash carrier is turned on, it will broadcast its BLE service and wait for Raspberry Pi board to connect to it.

On Raspberry Pi, we used the *bluepy* Python package to establish a BLE connection with the trash carrier. Whenever user presses the switch for trash disposal, it will determine whether the trash carrier is available, and send a pre-programmed rotation angle for each type of trash such that the trash carrier rotates accordingly.

D. Hierarchical Finite State Machine for Buckler

For the trash carrier, we used a hierarchical finite state machine (H-FSM) to model the environments and actions. As shown in Fig. 3, there are two high level states:

1) `Rotate State`, which accepts commands from the trash classifier and rotates the trash carrier to align the correct bin with the opening on the box. Whenever the trash carrier detects one of the bins is full, it will transit to the `Path Finding State`.
2) `Path Finding state`, which reads the vector information returned by CMU Pixy2 Camera to accomplish path finding to trash disposal and back. It will transit to `Rotate State` when the trash carrier is at its reset position.

The detailed implementation of the H-FSM is in Fig. 9, which can be found in the appendix. In the next section, we briefly discuss the modules that provide inputs/outputs to the H-FSM.

E. Buckler Peripheral Devices

1) CMU Pixy2 Camera on Serial Peripheral Interface (SPI): CMU Pixy2 Camera [4] captures images from its camera and recognizes multiple types of objects (color-connected components, lines, barcodes, etc.). In this project, we only use the line information, as demonstrated in Fig. 6.

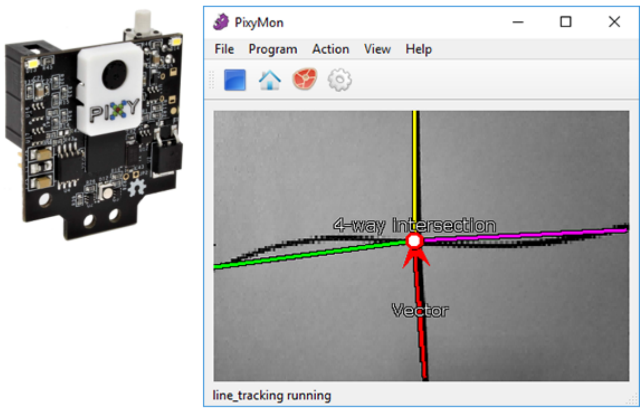


Fig. 6. CMU Pixy2 Camera is able to capture images of line-like objects and vectorize lines into vectors.

Pixy2 camera has its own microprocessor, and is able to process lines in camera frames at 60 frames per second. Four integers, representing the X/Y coordinates of the start/end points of each recognized line, are sent as a packet from Pixy2 camera and parsed by the trash carrier. We fine-tuned several parameters on Pixy2 camera such that it only recognizes tapes with a certain width and color, matching the tapes used by us. For each camera frame, Pixy2 camera also determines a "principle" component, which has the maximal recognition confidence. We only use this principle line vector and ignore all other non-principle line vectors.

Vectorized line information will be sent to the trash carrier via the SPI bus. Pixy2 camera acts as a SPI slave, and Buckler must actively perform SPI I/O to retrieve line vector information in each H-FSM cycle.

2) Ultrasonic Proximity Sensors: We used three ultrasonic proximity sensors to detect when any of the bins is full of trash. Three sensors are combined using logical or gate, which is implemented using NAND gate IC's. The input/output of ultrasonic sensors are connected with two GPIO pins on Buckler. We read ultrasonic sensor status every 1 second, and indicate the bin is full if 5 positives are read in a row.

3) Kobuki Driver: We used the Buckler library to initialize and control the Kobuki robot. To avoid abrupt accelerations during H-FSM transitions, we mix the current wheel speed v_i^t with the desired speed values v_i^* at each cycle to get the control speed values v_i^{t+1} which are then sent to Kobuki:

$$v_i^{t+1} = \alpha v_i^t + (1-\alpha) v_i^*, i = \{left, right\}, \tag{1}$$

where α is a mixing parameter and set to 0.7 in our project.

In the path finding state, we implemented a PID controller:

$$\begin{aligned} \delta x^t &= x^t - x_0, \\ v_{turn}^* &= K_p \delta x^t + K_d(\delta x^t - \delta x^{t-1}), \\ v_{left}^* &= v_{base} + v_{turn}^*, && (2) \\ v_{right}^* &= v_{base} - v_{turn}^*, && (3) \end{aligned}$$

where x^t is the variable to be controlled (x coordinate of the end point of the principle line vector returned by Pixy2 camera, normalized to $[0, 255.0]$); x_0 is the desired value (set to the middle of Pixy camera view $x_0 = 127.0$); δx^t is the error of the variable at time t; v_{base} is the base speed of the trash carrier when no turn is needed. Two PID control parameters are set to $K_p = 0.5$ and $K_d = 1.0$. Left and right wheel speed calculated from Eq. 2 and Eq. 3 are then put to Eq. 1 where the real wheel speed is computed and sent to Kobuki.

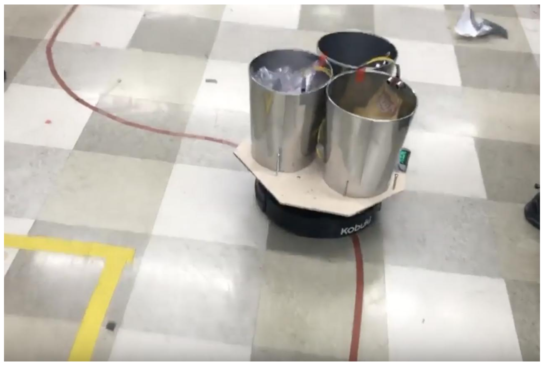


Fig. 7. The trash carrier performing path finding.

III. Course Concepts

1) Sensors and Actuators: We used a different variety of sensors in each module of the system: *Raspberry Pi Camera* captures image of trash for classify; *CMU Pixy2 Cam* detects colored taps on the ground to identify the position of the trash carrier; *Ultrasonic Proximity Sensor* detects bin fullness; *MPU9250 Gyroscope* on Buckler tracks rotation of Kobuki; *Kobuki* actuates driving and carrying all bins.

2) Networking: We used *Bluetooth Low Energy* for communication between Raspberry Pi the trash carrier. Rotation commands are sent from Raspberry Pi to Buckler through BLE such that Kobuki can align the specific bin.

3) Continuous Dynamics: We used *PID Feedback Controller* in the path finding module to ensure a robust control of the trash carrier. Tape vector detected by Pixy2 Camera is controlled to the center of view so Kobuki can follow the tape.

4) Finite-State Machines: We used *Hierarchical Finite State Machine* to incorporate the two major functions of Kobuki: Rotation and Path Finding. Futhermore, We used Yakindu as a designing tool for our state machine.

IV. System Evaluation and Results

A. Computer Vision

We evaluate the accuracy of our Computer Vision trash classification system. Our system attempts to classify an image to trash, recycling, compost, or no object/unknown, which defaults the trash bin to trash.

TABLE I
Trash Classification Evaluation

Metrics	No Object	Recycling	Compost	Trash	Total
Accuracy	71.43%	100%	100%	57.14%	71.43%
F1 Score	0.80	1.00	1.00	0.67	0.79

Our classification system is fairly accurate with a total classification accuracy of **71.43%**. However it should be noted our sample size is 7 objects found in the Embedded Systems Lab and is not statistically significant. Recycling and Compost are very accurate because they are distinct objects such as soda cans and apples. These objects are common and easy to be classified by a CV algorithm.

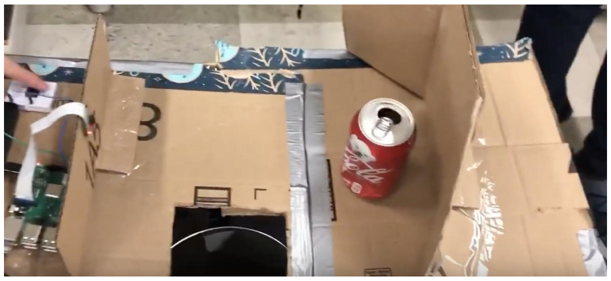


Fig. 8. Example of sorting a Coke can.

The hardest classes to classify are No Object and Trash. For No Object, other objects are misclassified as it because the background wall appears in many other objects. We were able to improve this by added a single line fix where when an object that is classified as nothing, check if it also matches at least two classes of another object. In that case, it is likely the object so we set the class to that object. For example, crumpled up aluminium foil exposes a lot of the wall and may be misclassified as no object. But the secondary check correctly classifies it as recycling. This improved the accuracy of our system in the demo, but was done after we evaluated our CV system.

Finally, trash is the biggest class of objects, making it easily confused with other objects that may fall into recycling or compost. We could improve our accuracy on trash, with a finer grain trash classification model.

B. Sensor Evaluations

Three of our sensors—the Buckler's MPU9250 gyroscope, the ultrasonic sensors on the bins, and the Pixy2 Camera—had inaccuracies embedded in them that required workarounds.

The gyroscope has an error of $\pm 10°$ per rotation, which can knock the bins out of alignment under the box's hole when trash gets inserted to the bin. This was fixed by calibrating the rotation through the help of the Pixy2 camera looking at the path tape each time the Kobuki rotates. This was done in such a way that every time the Kobuki rotation passes zero degrees, the trash carrier will rotate in one direction until Pixy2 Camera gets a glimpse of the tape in a specific location, as described in the `Reset_Align` state in the H-FSM in Fig. 9.

Given that we used small bins, the ultrasonic sensors attached on their upper edges were susceptible to picking up random sound reflections that made it seem like the bin was full when it was not. The solution to this was writing the sensor checking code in a way such that if the sensors were continuously full for around 5 seconds then it would initiate its driving to the garbage man.

Lastly, there was around a 10% error from the vector returned by the Pixy2 Camera when looking at the colored tape during path finding. The aforementioned PID controller was enough to drive along the path accurately even with this error embedded to the system.

V. What Worked and What Did Not

For the most part, the components to this project seamlessly fell in place and worked properly by the end. We were not forced to iterate much from our original vision. With that said, there were some aspects of this project we could have improved.

To begin with, the box that the Kobuki initially rotates under was far from ideal. While the purpose of this box was to demonstrate how a user could use the RecycleBot to easily dispose of his trash in the proper bin, the nature of its cardboard material jeopardized its structural integrity. There were times were it would fall on top of the Kobuki and prevent easy exit and entry. In an ideal situation, we would have demonstrated this with a more stable structure, such as a wood box.

Another issue with the RecycleBot was that after it did its initial path finding when a bin was full, it did not reenter the box with proper alignment. That is, it did not enter the box in such a way that allowed for continued use because the bins were no longer aligned in such a way that a user could dispose his trash. We are not sure why it does this, conjecturing that it may be due to unaccounted changes in the Kobuki's wheel speeds when it rotates after it initial path finding towards the garbage man. In an ideal scenario, the RecycleBot would be able to perpetually gather and dispose trash without the need for manual readjustment.

There were also issues with the Bluetooth connection when interfacing the Buckler with the Raspberry Pi. In particular, the connection between the two would break and we were forced to restart the Kobuki to reconnect them, possibly because it would move too far. Obviously, for the RecylceBot to work in any practical setting it should attempt to reconnect any communications like these at the very least.

Finally, during image classification, Google Cloud Vision does not return specific object classifications but instead returns a list of "noisy labels" that it believes are in the image. For example, when it took a picture of an apple it returned a list of labels such as "apple, red, fruit, sphere". To work around this, our code would count up how many labels corresponded to each bin and then classify it according to that.

VI. Conclusion

We designed and created RecycleBot, a smart trash bin to automatically sort and take out trash. Our system takes picture of trash, classifies it through Google Cloud Vision, and aligns the proper bin under a opening for a user to dispose. It also detects if a bin is full and performs path finding by following a tape on the ground to drive itself to a specific location. Experiments show that our system can successfully sort different trash and drive to the disposal position with robustness. Overall, RecycleBot is an effective solution for making trash, recycling, and compost disposal more practical and efficient.

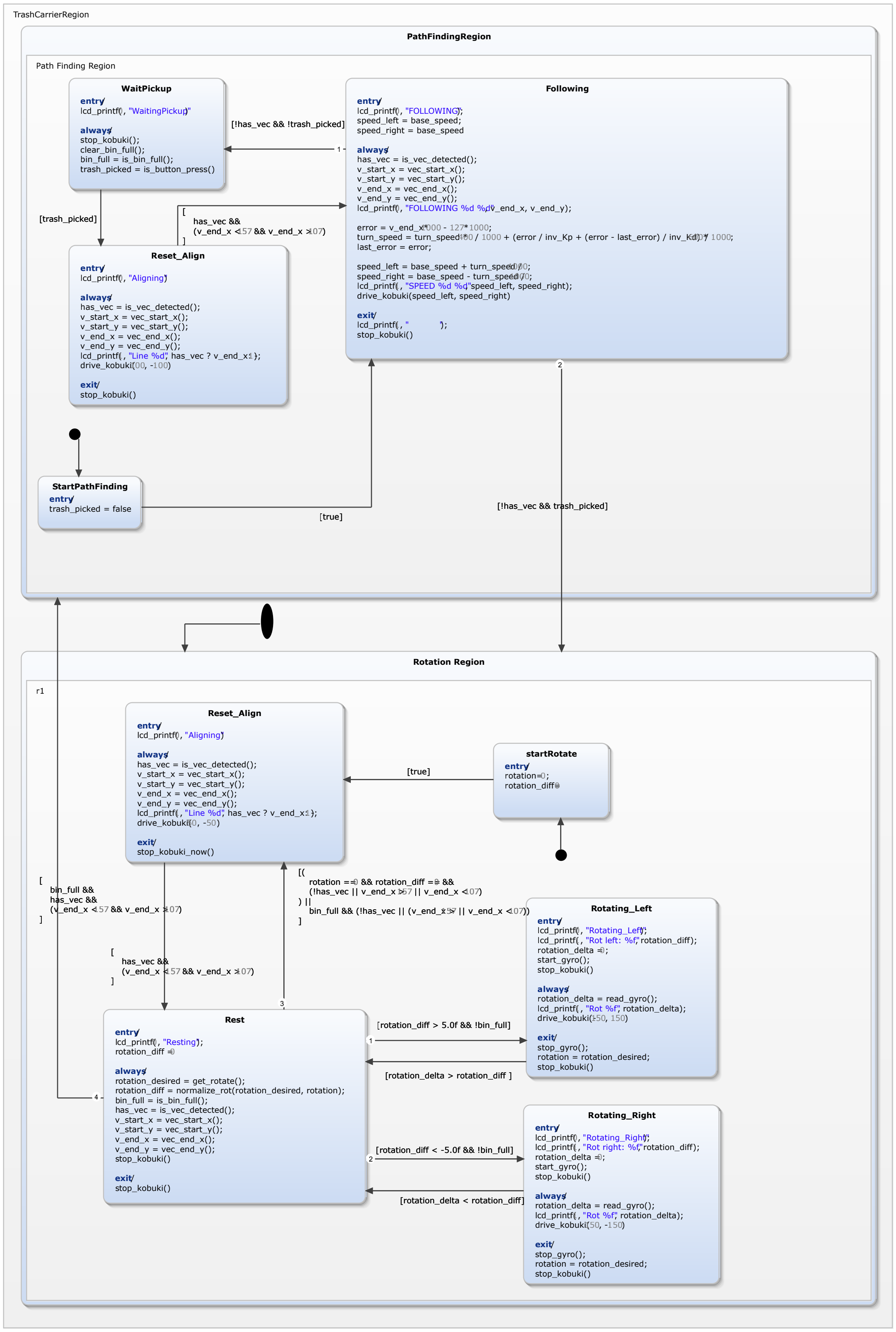
References

[1] Buckler Developers. "Berkeley Buckler." *GitHub*, 16 Dec. 2019, https://github.com/lab11/buckler.

[2] Google Cloud. "Vision AI, Derive Image Insights via ML." *Google*, 16 Dec. 2019, https://cloud.google.com/vision/.

[3] Dave Jones. "picamera — Picamera 1.13 Documentation." *readthedocs*, 16 Dec. 2019, https://picamera.readthedocs.io/en/release-1.13/

[4] Charmed Labs. "Pixy2 Index." *Pixy2 Documentation*, 16 Dec. 2019, https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:start



```
WaitPickup
entry/
lcd_printf(0, "WaitingPickup")

always/
stop_kobuki();
clear_bin_full();
bin_full = is_bin_full();
trash_picked = is_button_press()
```

```
Following
entry/
lcd_printf(0, "FOLLOWING");
speed_left = base_speed;
speed_right = base_speed

always/
has_vec = is_vec_detected();
v_start_x = vec_start_x();
v_start_y = vec_start_y();
v_end_x = vec_end_x();
v_end_y = vec_end_y();
lcd_printf(0, "FOLLOWING %d %d", v_end_x, v_end_y);

error = v_end_x*1000 - 127*1000;
turn_speed = turn_speed*400 / 1000 + (error / inv_Kp + (error - last_error) / inv_Kd) * 600 / 1000;
last_error = error;

speed_left = base_speed + turn_speed / 1000;
speed_right = base_speed - turn_speed / 1000;
lcd_printf(1, "SPEED %d %d", speed_left, speed_right);
drive_kobuki(speed_left, speed_right)

exit/
lcd_printf(1, "          ");
stop_kobuki()
```

```
Reset_Align
entry/
lcd_printf(0, "Aligning")

always/
has_vec = is_vec_detected();
v_start_x = vec_start_x();
v_start_y = vec_start_y();
v_end_x = vec_end_x();
v_end_y = vec_end_y();
lcd_printf(1, "Line %d", has_vec ? v_end_x: -1 );
drive_kobuki(100, -100)

exit/
stop_kobuki()
```

```
StartPathFinding
entry/
trash_picked = false
```

```
[!has_vec && !trash_picked]
[trash_picked]
[
    has_vec &&
    (v_end_x < 157 && v_end_x > 107)
]
[true]
[!has_vec && trash_picked]
```

```
Reset_Align
entry/
lcd_printf(0, "Aligning")

always/
has_vec = is_vec_detected();
v_start_x = vec_start_x();
v_start_y = vec_start_y();
v_end_x = vec_end_x();
v_end_y = vec_end_y();
lcd_printf(1, "Line %d", has_vec ? v_end_x: -1 );
drive_kobuki(50, -50)

exit/
stop_kobuki_now()
```

```
startRotate
entry/
rotation=0;
rotation_diff=0
```

```
Rotating_Left
entry/
lcd_printf(0, "Rotating_Left");
lcd_printf(1, "Rot left: %f", rotation_diff);
rotation_delta = 0;
start_gyro();
stop_kobuki()

always/
rotation_delta = read_gyro();
lcd_printf(1, "Rot %f", rotation_delta);
drive_kobuki(-150, 150)

exit/
stop_gyro();
rotation = rotation_desired;
stop_kobuki()
```

```
Rest
entry/
lcd_printf(0, "Resting");
rotation_diff = 0

always/
rotation_desired = get_rotate();
rotation_diff = normalize_rot(rotation_desired, rotation);
bin_full = is_bin_full();
has_vec = is_vec_detected();
v_start_x = vec_start_x();
v_start_y = vec_start_y();
v_end_x = vec_end_x();
v_end_y = vec_end_y();
stop_kobuki()

exit/
stop_kobuki()
```

```
Rotating_Right
entry/
lcd_printf(0, "Rotating_Right");
lcd_printf(1, "Rot right: %f", rotation_diff);
rotation_delta = 0;
start_gyro();
stop_kobuki()

always/
rotation_delta = read_gyro();
lcd_printf(1, "Rot %f", rotation_delta);
drive_kobuki(50, -150)

exit/
stop_gyro();
rotation = rotation_desired;
stop_kobuki()
```

```
[true]
[
    bin_full &&
    has_vec &&
    (v_end_x < 157 && v_end_x > 107)
]
[
    has_vec &&
    (v_end_x < 157 && v_end_x > 107)
]
[(
    rotation == 0 && rotation_diff == 0 &&
    (!has_vec || v_end_x > 157 || v_end_x < 107)
) ||
    bin_full && (!has_vec || (v_end_x > 157 || v_end_x < 107))
]
[rotation_diff > 5.0f && !bin_full]
[rotation_delta > rotation_diff ]
[rotation_diff < -5.0f && !bin_full]
[rotation_delta < rotation_diff]
```

Fig. 9. The hierarchical finite state machine for modeling the trash carrier environments and actions.