Smart Trash Can Final Report

Author: Ruohai Ge, Xingsheng Wang and Zheng Xu

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—We are developing a smart trash can that would approach and catch the trash within a certain area instead of people walking towards it to throw their trash. The trash can is located near the corner of the room and can cover a circular area of a 1m radius. We use a depth camera to detect and track the object and a mobile base to move the trash can. A computer with Bluetooth is used to process camera information and send position information to the mobile base.**

***Index Terms*—smart trash can, object detection, Kalman filter, customized YOLOv3 model, PID, dead reckoning, depth camera, mecanum wheel.**

I. Introduction

Have you ever been annoyed of walking towards the trash can to throw your trash. In addition, if one is unable to move easily, the simple task of throwing trash might be a painstaking process. Therefore, we introduce the smart trash can to make throwing trash easier. Our trash can operates within a 1-meter radius circular area near a corner of the room. Users throw their trash to the 1-meter radius circular area and the trash can reaches the landing location of the trash (~1s) before the trash lands and catches the trash on its own, without users leaving their current locations. The trash can returns to the center of the operation area and waits for the next trash. Besides, our project introduces a more hygiene approach to throwing trash by reducing contact with the trash can.

The smart trash cans available in the markets have very different approaches. Most products can only sense a person nearby and open the lid for the person who wants to throw his/her trash. There are no similar products currently and there is only a person posting a video about this project which he achieved a success rate around 20%.

II. Design Requirements

We have split our requirements into the following categories: hardware and setup, users' behaviors, hardware accuracy, software accuracy and result. Hardware and setup includes trash can's position and size, trash's size, camera's position and the movement range. Users' behaviors define users' throwing patterns and ranges. Hardware accuracy and software accuracy ensure the trash can is able to catch trash inside the 1m movement range within 1 second. Result defines the overall success result and expected behavior.

A. Hardware and Setup

Our hardware setup originates from the setup of common thrash cans at home or office. The trash can is placed near the corner between two walls since this is the usual place to put a trash can. Trash can moves in the range of 1m starting from the center of the 1m movement area (30 cm from both walls). The trash can has an upper 33cm diameter, lower 23cm diameter, 42cm height, and around 2 pounds, which is similar to trash cans commonly used in the bedroom or office. The two walls are 3 meters tall which are the normal floor heights. The camera is mounted on the wall, at least 2 meters above the trash can, which has an unobstructed view of the trash once the trash leaves the hand. The entire physical setup is displayed in Fig. 1. Trash is a toy softball, which has a 9.6cm diameter and a mass around 43g. This softball has normal trash size and mass, which assemblies paper balls and empty tin cans (common trash in bedroom or office). The whole system uses a computer with a graphics processing unit.

Fig. 1. Physical Setup

B. Users' Behaviors

Users can throw from a region between 1-3 meters away from the corner of the wall. Since the bedroom is usually 3m*5m, there can be at most a 3m*3m free space. Thus, 3 meters is the maximum range. Within a 1m range, people can

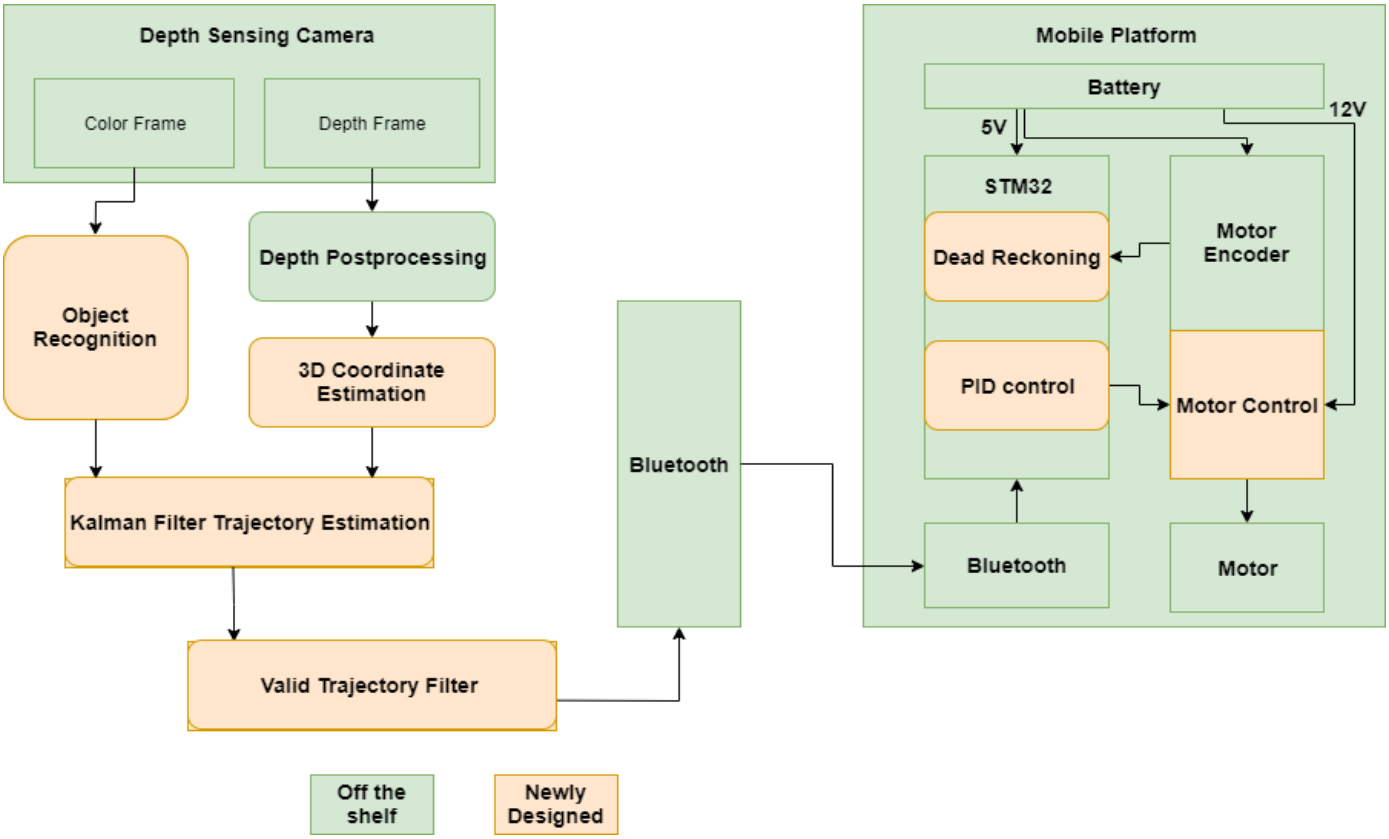

Fig. 2. Physical Setup

easily throw the trash inside the trashcan, which makes this project useless. So, 1 meter is the minimum range. Users throwing pattern is derived from normal patterns of people throwing garbage. Users are required to throw the trash under the height of the wall (2m), throw with speed around 5m/s (+-0.5m/s), throw toward the 1m movement range of the trash can. Throwing hand's position must be above 1 meter and throw 1 trash ball at a time, which means users start throwing only after trash can returns to its initial position (30 cm from both walls).

C. *Hardware Accuracy*

For hardware accuracy, we have three main criteria to fulfill in order to achieve the project's goal. The Bluetooth information must be received in 0.2 seconds, the mobile base must move to the designated position within 5cm and able to reach this position within 0.5 seconds. The 0.2 seconds of the Bluetooth communication and the 0.5 seconds of the mobile platform movement ensures that the trash can be caught in 1 second. Since the trash can has a diameter of 33cm and the trash is 10cm diameter, the 5cm position error is still tolerable. For testing, timestamps are used to test the time of Bluetooth communication. By giving arbitrary points in the 1m movement range, we can test the mobile base's position accuracy and movement time.

D. *Software Accuracy*

For software accuracy, we have four main criteria to fulfill in order to achieve the project's goal. Our system could recognize the trash within 0.2 seconds and with a false-positive rate less than 5% and can be tracked at 20fps. 20fps is the minimum requirement for tracking to finish under 0.4 seconds, which will be further explained in the system description. These will be tested by using video sequences with bounding boxes. These two requirements need to be fulfilled for the 1 second runtime goal. 3D coordinates transformation should have an accuracy of 5cm, and trajectory estimation should have an accuracy of 10cm. Trajectory estimation must give its initial landing position within 0.2 seconds and final landing points within 0.5 seconds, which will be further explained in the system description. For 3D coordinates accuracy, comparing calculated and real coordinates of a fixed position object. Trajectory estimation is tested by comparing color-labeled predicted trajectory and real trajectory in a video of a moving object. These two requirements need to be met for the goal of successfully catching the trash.

E. *Result*

Trash can should catch the trash at an 80% success rate. Success means the ball completely stays inside the trash can. This success rate creates a relatively good user experience and the failure does not cause any safety issue. Furthermore, a similar previous project only has a 10%-20% success rate. For the expected behavior, the trash can should never move out of the designated area (1m range from the corner of the wall) and return to the center position (30 cm from both walls) after the trash lands.

III. Architecture and/or Principle of Operation

The general architecture is depicted in Fig. 2 and the zoomed-in architecture diagrams will be displayed in the System description part.

The entire system is split into two parts, the computer vision part and the mobile platform part. The computer vision part oversees detecting the object and deciding its landing position. The mobile platform part controls and records the movement of the trashcan and guides the trashcan to the landing position. The reason why these two parts are separate is that the computer vision part requires much computation power that is hard to achieve on a small, light moving platform, and a bloated moving platform will add complexity to the project. The depth camera provides both RGB information and depth information. We use the RGB information to identify and track the trash and use the depth information to transform pixel location to a real-world position. We first identify the object in the color frame. After each detection phase, the program computes its real-world 3D coordinates. Before inputting the coordinates into the trajectory prediction, we first do some checks on the current and previous few positions of the detected object to remove noises and make sure that the object is moving. The trajectory estimation will continuously output prediction of landing point instead of outputting one at the end. At the first few iterations, the trajectory estimation outputs a very coarse estimation, as there are more inputs, the estimation becomes more accurate. The reason for this pipeline design is that we have limited time to move our platform, and the sooner we guide the platform to a rough direction, the more time we have for finer control of the moving platform. Each estimation output is transmitted via Bluetooth to the moving platform, where the dead reckoning algorithm estimates where the platform currently is and the PID control guides the platform to its goal position. There is no knowledge of the position of the moving platform in the computer vision module.

The software and hardware modules that we use are listed below.

A. Python

We are using Python as our programming language for the computer vision module because we use OpenCV, PyTorch and Realsense SDK, which are all available on Python.

B. OpenCV

We choose OpenCV because most of the computer algorithms that we use are implemented in OpenCV, and the runtime is much better than self-implemented algorithms.

C. YOLOv3 in PyTorch

YOLOv3 is a lightweight machine learning object detection algorithm. This algorithm works for us because its object detection indicates the location of the object in the image and the speed of the algorithm is very fast. We choose the implementation of this algorithm in PyTorch because to achieve real-time speed we need GPU support, and PyTorch provides an easy interface to use GPU in Python.

D. Intel Realsense SDK

This is a library for using the Realsense depth cameras, which provides functionalities as accessing depth frames, postprocessing depth frames and transforming from depth frames to 3D coordinates. All of these functionalities are useful when we need to extract 3D position of the object tracked in the RGB frame.

IV. Design Trade Studies

A. GMG Background Subtraction vs Customized YOLOv3 Object Detection

Object detection aims at removing the background part of the image and extracting the trash out of the image. GMG background subtraction's principle is to detect moving objects inside a frame. Since this method does not involve any learning algorithm, it can ensure the vision pipeline to run at around 20fps. However, the trash normally is not the only moving object inside frames. Moving hands and shaking bodies are all common noises that cannot be easily filtered out. Thus, the false-positive rate of the GMG background subtraction is normally around 20%. The false-positive even increases to 30% when more people appear inside the frame. On the other hand, the customized YOLOv3 model for this specific type of trash has a false-positive rate smaller than 5%. The model is very lightweight and is customized using a specific resolution balanced between accuracy and speed. With the help of the graphics processing unit, the vision pipeline can still run at around 20fps. With very little noise, the ball can be detected in only 0.08 seconds (around 2 frames). With good accuracy, less noise and acceptable fps, we finally choose customized YOLOv3 object detection over GMG background subtraction.

TABLE I. Comparison between GMG and YOLOv3

Metric	*GMG*	*YOLOv3*
Frames per second	20	20
False-positive rate	20%	5%
Trash recognition speed	0.2s	0.08s

B. Kalman Filter Vs Reinforcement Learning

The learning method is a very popular method to predict and estimate trajectories. The learning method typically provides a very accurate estimated trajectory and is more adapted to general cases, which can work under many different noises. However, the learning method is too slow for this project. From the paper Regularizing Neural Networks for Future Trajectory Prediction via Inverse Reinforcement Learning, it states that each learning method observed 8 frames (3.2sec) and predicted the next 12 frames (4.8sec) [1]. This is too slow to achieve the requirement of catching the trash in 1 second. Kalman filter is the traditional way of trajectory estimation and prediction. It works relatively accurate under many cases with noises. Although it provides less accurate results and works with less general cases than the learning method does, our project has error tolerance and performs under a relatively stable indoor environment (less noise). More importantly, this method is fast enough to meet our runtime requirement, which is just matrix multiplications that can be finished within 0.1 seconds.

C. Bluetooth Vs Wi-Fi

While Wi-Fi can cover a greater distance and provide faster communications between devices, there are timing issues on

other parts of the system. As stated before, the trash will hit the ground around 1s after leaving the user's hand and there is not much area that the trash can is able to cover. Therefore, we have limited the trash can's area of operation to a circle 1m radius from a corner of the room. Bluetooth is more than capable of covering this area. In addition, we timed the transmission time of Bluetooth connection and it is within 0.1s, which is well within our limits. Moreover, the Bluetooth module uses a serial interface to communicate with the embedded board mounted on the mobile base so it easier to implement than a Wi-Fi module, which will reduce the complexity of the project.

D. Mecanum Wheel Vs Regular Wheels/Track

As our project is timing critical, we have to make the trash can move to the desired location as fast as possible to catch the trash in time. Therefore, we need to reduce the amount of motion needed to reach to the destination. Regular wheels will result in the rotation of the mobile base to align with the final destination and then move towards the landing location of the trash. The rotation will cost precious time as it needs adjustments to achieve the desired rotation degree. Using track also requires the mobile base to rotate before moving in a straight line to the desired location. Both solutions involve the rotatory motion of the platform which takes time. However, Mecanum wheels will allow the platform to move straight to the destination without rotation. Hence, time is saved, and we can leave more time for small adjustments of the trash can's final position.

TABLE II. COMPARISON BETWEEN SIDE TRANSLATION AND ROTATION WITH TRANSLATION

Metric	***Side Translation***	***Rotation with Translation***
Average time to reach a designated points inside the 1m movement range	0.4s	1.2s
Average deviation/error from a designated points inside the 1m movement range	4.1cm	4cm

E. Dead Reckoning Vs Distance Sensor Triangulation/Visual or Depth Sensor

The distance sensor triangulation method using WIFI or Bluetooth is less accurate than dead reckoning. Visual or depth sensor-based localization has too much complexity for a moving platform and is difficult to achieve centimeter-level accuracy without additional sensors such as the Inertial measurement unit (IMU). By comparing to these other methods, dead reckoning is fast, simple to implemented and requires no additional sensors. It might not be as accurate as other localization methods. However, some drifts and errors are tolerable for our project since the trash can does not need to be exactly at the designated coordinates. Its upper diameter is 20cm longer than the trash diameter, so drifts around 5cm is acceptable. Furthermore, after each throw, the trash can returns to its initial position and resets the encoder values. So, we don't need to worry about cumulative errors.

F. PID Vs Linear Quadratic Regulator

Linear Quadratic Regulator does not provide significant improvement in the scenario of this project. It's very complicated which requires predefined trajectories used for optimization. It also requires a linear dynamic model which is not what we want. We only want simple position control instead of torque control. Thus, PID is the best control method for our project. It is simple to implement, fast and also reliable. We focus more on tuning position instead of torque and do not need to create any dynamic models. The only problem of PID is that it needs to be tuned every time when a new environment is introduced. However, our project performs under a relatively stable indoor environment with a similar floor material. Thus, after the initial tuning, we only need to make small changes.

V. System Description

A. Image Preprocessing

Image preprocessing takes the raw image from the depth camera and conducts some basic operations like edge-preserving blurring and region of interest selection to reduce the size and noise of the RGB input.

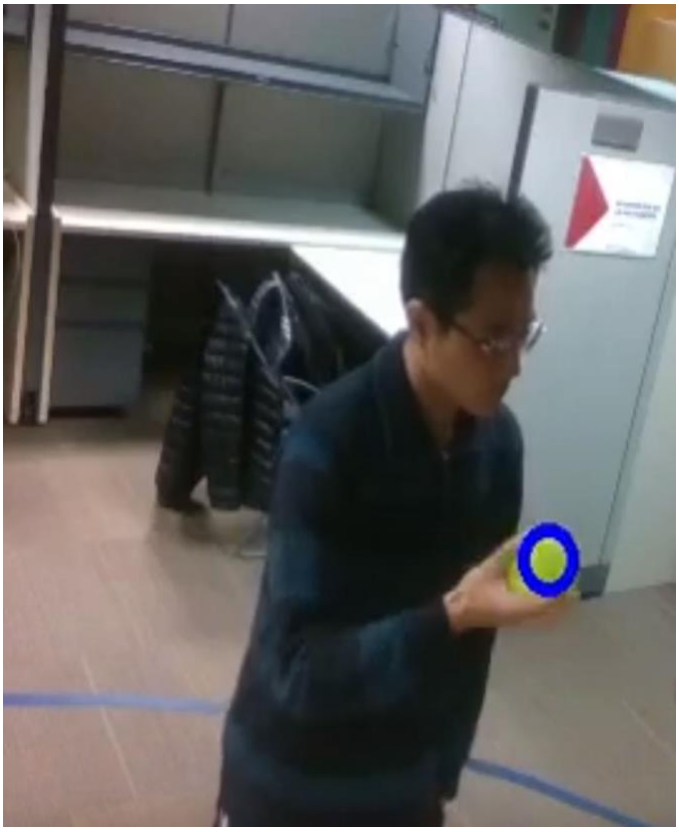

Fig. 3. Trash Detection Effect

B. Object Detection

We use YOLOv3 to detect the object. Our implementation has a minimum requirement of 4GB of RAM and graphics processing unit (GPU) support. Our solution runs on a laptop with 8 core CPU with 8 GB of RAM and GPU support. The

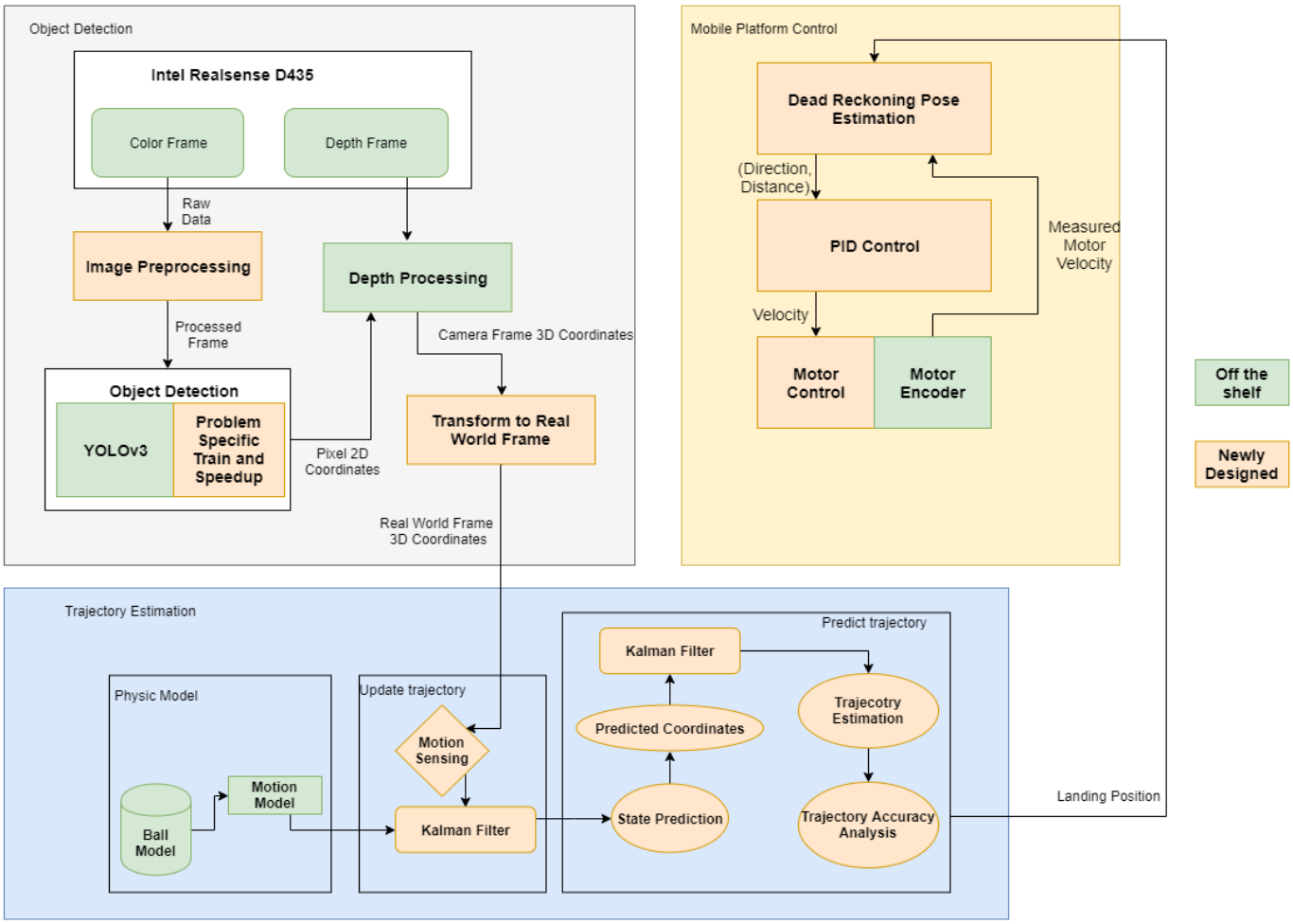

Fig. 4. Software Diagram

advantage of using YOLOv3 is that it is more accurate compared to the conventional vision method without sacrificing speed. According to our experiments, the most fined tuned object detection in traditional vision method (background removal, shape detection) achieves around 90% success rate and 20% false positive. The YOLOv3 network that we specifically trained for the object reaches a 95% success rate with less than 1% false positive. However, the conventional YOLOv3 network is not accurate enough in detecting object position and it is not fast enough for our real-time use case. There are a couple of changes that we made to the network to make sure that we achieve higher accuracy and more than 50% increase in speed. First, we reduced the input size to a minimum without affecting accuracy to improve speed. Second, we reduced the detection class to one and we only require the center position of the object so that the inaccuracy in the bounding box output can be reduced. In the end, we achieved the position accuracy to within 3 pixels and the inference speed is around 35ms. Fig. 3 shows the trash detection effect.

C. Depth Transformation

Only pixel location of the object is not enough to calculate the landing point in the real world. In order to transform pixel location into real-world coordinates, we need to use the depth frame generated in the depth camera. The depth frame from the the camera has a lot of fluctuations and has pixels where the camera cannot compute its depth. We will use some postprocessing included in the Realsense SDK to generate a smoother, more stable output. For example, the temporal filter smooths each pixel's depth value by looking at its previous value, and the spatial filter smooths each pixel's value by looking at its surrounding pixels. After post-processing, we use the 2D pixel position to get a 3D position in the camera frame. We measured the transform from camera frame to a real-world frame using preexisting tools with AR tags. Fig. 5. shows the setup and process of transform measurement. In the end, we managed to get the real-world coordinates of the object with an accuracy of around 5cm in each dimension.

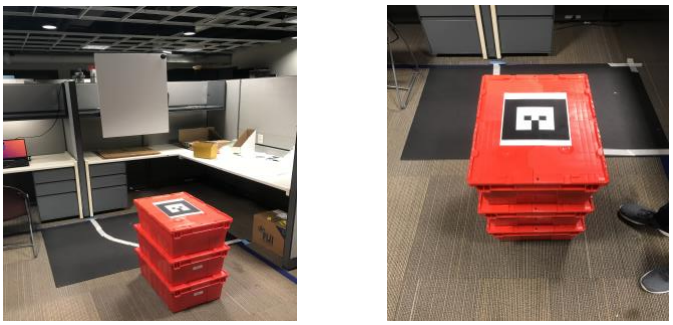

Fig. 5. Transform using AR Tags

D. Kalman Filter

We use the Kalman Filter for trajectory estimation and prediction. We use the classical physics motion model to estimate the motion of the ball. x, y, z, v_x, v_y and v_z mean the current state of the ball and $x^{'}$, $y^{'}$, $z^{'}$, $v_X{}^{'}$, $v_y{}^{'}$, $v_z{}^{'}$ mean the next state of the ball.

$$v_X{}^{'} = v_x + a_x * \Delta t \quad (1)$$

$$v_y{}^{'} = v_y + a_y * \Delta t \quad (2)$$

$$v_z{}^{'} = v_z + a_z * \Delta t \quad (3)$$

$$x^{'} = x + v_x * \Delta t + \frac{1}{2} * a_x * (\Delta t)^2 \quad (4)$$

$$y^{'} = y + v_y * \Delta t + \frac{1}{2} * a_y * (\Delta t)^2 \quad (5)$$

$$z^{'} = z + v_z * \Delta t + \frac{1}{2} * a_z * (\Delta t)^2 \quad (6)$$

Based on this motion model of the ball and the algorithm introduced from page 36 to 44 of the paper Analisis de eventos deportivos mediante vision artificial [3]. The following equations show the algorithm design of our Kalman Filter. F is the dynamic system model and B is the acceleration model. For the acceleration matrix, we assume that the ball only has a downward acceleration and does not have acceleration in x and y direction.

$$F = \begin{matrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} \qquad State = \begin{matrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{matrix}$$

$$B = \begin{matrix} \frac{1}{2} * (\Delta t)^2 & 0 & 0 \\ 0 & \frac{1}{2} * (\Delta t)^2 & 0 \\ 0 & 0 & \frac{1}{2} * (\Delta t)^2 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{matrix} \qquad a = \begin{matrix} 0 \\ 0 \\ -9.8 \end{matrix}$$

Based on the (1) to (6), we can get the predicted next state using the current state without observation.

$$\text{state\_predict} = F * \text{state} + B * a \quad (7)$$

Then we calculate the predicted covariance without observation, S is the noise of the motion model.

$$\text{covariance\_predict} = F * \text{covariance} * F^T + S \quad (8)$$

Then we calculate the difference between observation and prediction calculated in (7) if the observation is provided. H is the Kalman filter observation model and we only care about the position in this case. If the observation is not provided, the state_predict and covariance_predict will be set as the new_state and new_covariance.

$$H = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$

$$\varepsilon = observation - H * state\_predict \quad (8)$$

K is the Kalman gain and R represents the observation noise. We set the covariance_predict as c in (9) for better readability.

$$K = c * H^T * \left((H * c * H^T + R)\right)^{-1} \quad (9)$$

Finally, we can get the estimated new_state and new_covariance based on observation, state_predict and covariance_predict.

$$new\_state = state\_predict + K * \varepsilon \quad (10)$$

$$new\_covariance = (I - K * H) * \text{covariance\_predict} \quad (11)$$

Furthermore, we design to continuously estimate trajectories and predict landing points. The system then can update estimation and prediction based on new tracking points.

a. Kalman filter predicts the second point based on the first tracked coordinates
b. Predicts the third point based on the estimated second coordinates
c. Then creates the entire estimated trajectory
d. Kalman filter then receives the second tracked point and repeat b-d to update the new trajectory

The system sends out the first predicted land point within 0.2 seconds and the last predicted land points within 0.5 seconds. In this way, Trash can moves to a possible point first and then adjusts to the final landing point based on more accurate predicted landing points based on more observation, which increases the accuracy and also enables the trash can to start early to save time.

E. Landing Points Prediction

The following algorithm block shows the logical flow behind the software module displayed in Fig. 4. It explains how coordinates are sent to the Kalman Filter, when the Kalman filter is reset, how to filter out noises and how to validate a parabolic trajectory and out of bound landing points.

Algorithm 1 Landing Point Prediction

```
procedure LANDINGPREDICTION
init:
    fail_counter ← 0
    disappear_counter ← 0
    last_prediction ← None
    last_position ← None
loop:
    pixel2D ← YOLOv3 detection
    if no detection then
        disappear_counter ← disappear_counter + 1
        if disappear_counter > 2 then
            KALMAN_RESET()
            disappear_counter ← 0
        goto loop.
    position3D ← transform(pixel2D)
    if CLOSE(position3D, last_position) then
        KALMAN_RESET()
        goto loop.
    last_location ← position3D
    prediction ← KALMAN_PREDICT(position3D)
    if INCONSISTENT_POINT(last_prediction, prediction) then
        fail_counter ← disapear_counter + 1
        if disappear_counter > 4 then
            KALMAN_RESET()
    else
        SEND_COORDINATES(prediction)
    last_prediction ← prediction
    goto loop.
```

Fig. 6. Visual Recognition Pipeline

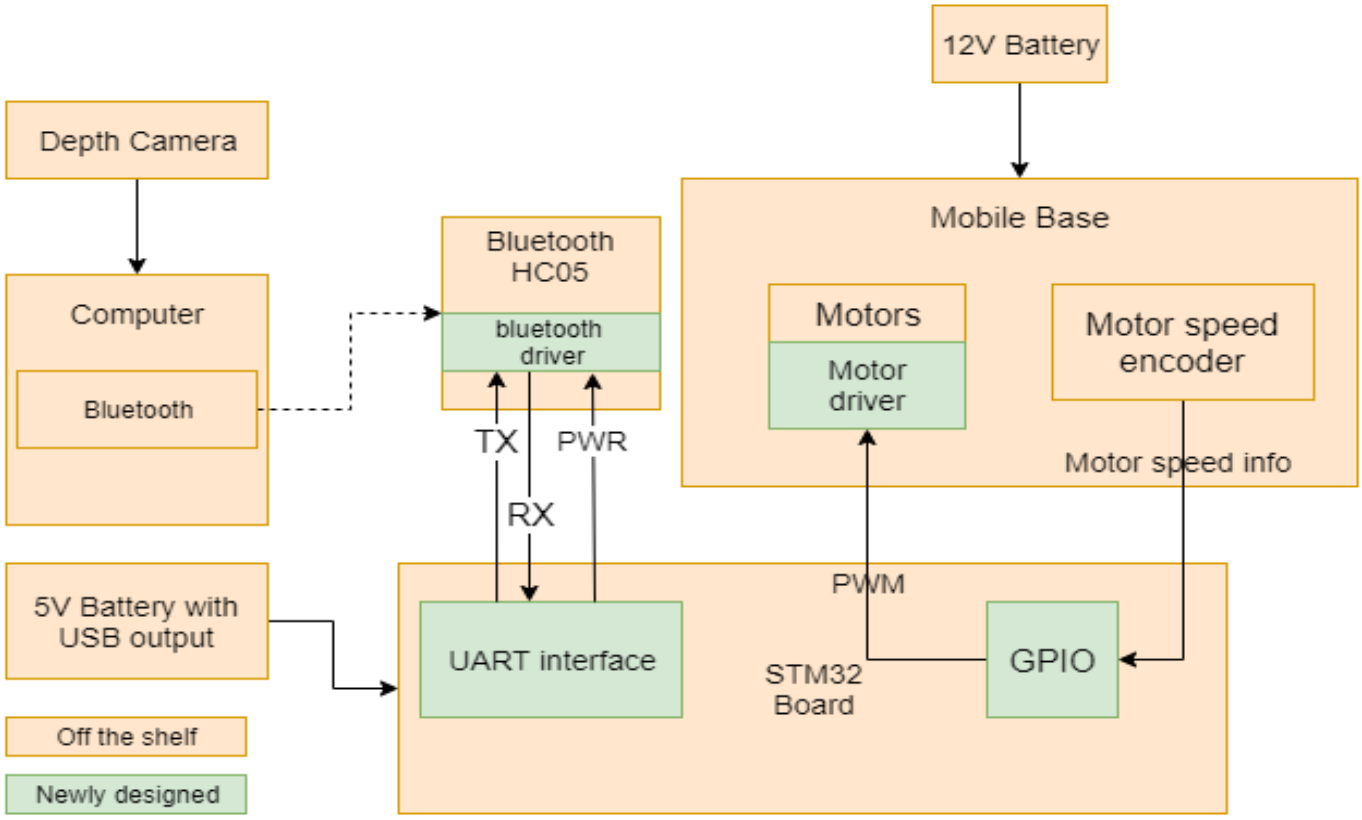

Fig. 7. Hardware Diagram

Fig. 8. Mobile Platfrom

F. Mobile Trash Can

The mobile trash can system includes a trash can, a mobile base with four motors, an embedded board, a Bluetooth module (HC-05) and a battery power supply. The embedded board is mounted on the top of the platform and the battery is mounted beneath the platform. There is an elevated platform 6cm above the mobile platform and the trash can is mounted on the elevated platform. The embedded board is an STM32 board with motor control connectors and Bluetooth module connectors. Details can be seen in Fig. 8. The Bluetooth module uses a serial interface to communicate with the embedded board. An interrupt will be triggered on the board and the position info will be saved in a queue on the board. There are four motor pin connectors on the board for the four motors on the mobile base. The board will send a PWM signal to control the motors and the speed encoders on the motors will send back the speed information for dead reckoning. Proportional-Integral-Derivative (PID) control is used to adjust the position of the mobile platform. The battery has a capacity of 3000mAh and a 12V output that powers both the motor and the board. The board takes in 12V and splits out a 5V to power the board and a 12V to power the motors. Every 5ms, there will be an interrupt and the board will check its queue to see if there is information sent via Bluetooth. It will update the platform's desired location based on the information in the queue. Speed information feedback would be transmitted to the embedded board from the motors. The embedded board will then adjust the PWM signal with PID control to move the trash can to the desired location. Hardware general diagram is displayed in Fig. 7.

G. Dead Reckoning

The dead reckoning program sets the center of the 1m movement area (30 cm from both walls) where the trashcan normally stays at as origin point. Once the trashcan starts working, the dead reckoning will always be running. At each iteration, the encoder updates the new motor motion to dead reckoning, and the program will use the same control algorithm used in Mecanum wheel control to estimate the current position of the platform. The control formula for the mecanum wheel platform is represented in Fig. 9. The dead reckoning algorithm performs well during the test. The maximum deviation from a designated point within the 1m movement range is only 4.1cm.

For a more detailed comparison, the mobile platform in total has three movements: moving forward and backward in a straight line, moving sideways and moving diagonally. Moving forward and backward in a straight line produces the best result since all four wheels are rotating in the same direction. It usually only produces an error under 1cm. Moving sideways and moving diagonally do not perform as well as moving forward and moving backward in a straight line. Since moving sideways requires different rotation directions of the four wheels and only two wheels need to rotate for moving diagonally.

$$v_{\omega 1} = v_{t_y} - v_{t_x} + \omega(a+b)$$
$$v_{\omega 2} = v_{t_y} + v_{t_x} - \omega(a+b)$$
$$v_{\omega 3} = v_{t_y} - v_{t_x} - \omega(a+b)$$
$$v_{\omega 4} = v_{t_y} + v_{t_x} + \omega(a+b)$$

v_{ω}: speed of Mecanum wheel

v_{t_x}: speed of vehicle in x-axis

v_{t_y}: speed of vehicle in y-axis

ω: angular velocity of vehicle

a, b: the length in x-axis and y-axis between the center point of vehicle and the center point of Mecanum wheel

Fig. 9. Mecanum Wheel Control [4]

H. PID control

PID control runs at the same time as dead reckoning. The position PID is implemented for our project. Every time the dead reckoning returns an estimated position, the PID will calculate a new error from the target encoder value with respect to the current encoder value, a derivative of the error (the difference between this error and the last error) and an integration of all previous errors. A new voltage for each of the 4 wheels will be calculated and sent to the motors. For the PID control, we spend some time to tune the parameters. Since we want to balance between speed and accuracy. Sudden acceleration is not optimal for our mobile base platform with the mecanum wheel since it might introduce some unbalanced friction or force which affects the car's movement. So, after careful tuning and thinking, we decided to make P = 6, I = 0.01 and D = 0. We need a relatively big P since we want the mobile platform with an initial high speed in order to reach the desired landing points within 0.5 seconds. Also, the P cannot be too big since it might introduce an unbalanced force as discussed before. D is set to 0 since our normal movement distance is very short (around 20cm -30cm). Nearly no oscillation or overshoot is going to happen. Last but not least, I term is introduced due to the reason that our mobile platform needs to run continuously. So, error culmination is unavoidable due to potential drift. The I term could mitigate this culmination error. Also, The I term cannot be too big since it might cause unwanted overshoot.

VI. Project Management

A. Schedule

There are no major changes to the schedule. We have been following the schedule strictly. At one point our trajectory estimation was completed ahead of time so we added another task in between. The task was to use machine learning to improve our object detection since we are using green softballs as our trash. See the appendix for our schedule details.

B. Team Member Responsibilities

Ruohai Ge is in charge of the trajectory estimation and landing position generation of the trash, YOLOv3 model training and sending the information to the mobile base. Zheng Xu is in charge of image filtering and the detection, tracking of the trash and coordinate transformation. Xingsheng Wang is in charge of implementing the drivers for the motors with PID control, the Bluetooth module, the communication protocol and setting up the whole mobile platform.

C. Budget

Please refer to TABLE III for a detailed description of the parts list. The parts that we need are within the budget line and there is still budget left in case any component fails, or additional components are needed. Most of the components are robust and the parts that may break are of low cost. The STM32 embedded boards were not that well manufactured and due to a short circuit two of the three boards that we have purchased were broken. Standoffs were added and there were no issues since then.

TABLE III. Budget and Parts

Item	Price
Moebius omni wheel mobile base	$148.72
Intel Realsense D435 depth camera	$198.91
STM32 embedded board	$28.14 * 3
TalentCell rechargeable battery 3000mAh 12V output	$25.85
Aideepen HC-05 bluetooth module	$27.8
Trash Can	$21.39
6ft long USB cable	$13.77
Softballs (used as trash)	$21.19
Cardboard as floor	$12.83
Wall mount for depth camera	$13.9
Wooden elevated platform (laser cut from spare wood)	$0.00
Standoff for platform and embedded board (from spare parts in lab)	$0.00
Total	$568.68

D. Risk Management

The first risk is related to the detection noise. The original GMG background subtraction uses the motion detection algorithm. However, there are too many things moving around inside one frame and it is impossible to find out which one is the trash. Furthermore, it is not optimal to force users not to move. After careful research on multiple different detection, tracking, motion capture methods, we found out that a customized YOLOv3 model is a valid alternative to minimize

the detection noises and reduces our detection false positive rate. YOLOv3 balances accuracy, speed and complexity perfectly, which helps our whole object detection part.

The second risk happened that just after we switched to YOLOv3. The model is not quick enough and the OpenCV structure does not provide any CUDA acceleration. The accuracy is increased but we cannot accept only detecting for 4-5 frames. After carefully searching online, thanks to the link at [5], we found a modified PyTorch version of the YOLOv3 model. PyTorch has great compatibility with the CUDA and GPU acceleration. With the help of this new model, we are able to reach the fps rate of 15. However, 15 fps is still not enough for our requirements. We then figured out that we can change the resolution of the picture that feeds into the model. Since lower resolution pictures take less time for the model to inference. We then carefully balanced between the accuracy and speed and decided to use the resolution of 640*360. The fps of the new vision pipeline finally reaches around 20fps.

The third risk that we have encountered was that the mobile platform's motion was not desirable. There were drifts when the platform moves that causes the position of the platform to be off from the indicated position. The problem was with the surface that the mecanum wheels operated on. Previously we tested the platform on the carpet and there were undesired drifting motions. We tested some other material including taped surface, acrylic and cardboard. In the end, we found that cardboard performed the best and used the cardboard as the "floor" in our scenario. Another factor that causes the drift was the PID control. The initial acceleration of the platform was too fast and in diagonal motions not all wheels are moving. The sudden acceleration then causes the platform to rotate a little which will affect the final position of the mobile platform. We tuned the PID parameters to reduce the acceleration and the motion of the mobile platform was greatly improved.

The last risk was also related to the platform motion. After some tests of ball catching, the motion performance of the wheels was dramatically worse. Sometimes the wheels spin but the platform was barely moving. We weighed the four wheels on four scales and discovered that the weight difference of the four wheels was large. This meant that there was a problem with the center of gravity of the whole system that affected the weight distribution and caused the wheels to have different weight distributed. A further investigation of the issue we found that this was due to the ball not being caught by the trash can and the wheels were hit by the trash. This causes a slight change in the angle of the mount of the wheels and hence the imbalance weight distribution. We solved this problem by recalibrating the wheels, but this problem repeatedly appears. There would be too much change if we switched to another platform and used different mounts so in the end, we decided to use weight scales to recalibrate the wheels when the weight distribution was incorrect.

VII. Related work

There aren't many competitors in the current market. We did not find any trash can products that will be able to move to catch the trash thrown by the user. The most similar ones are the ones that would open the lid automatically when you want to throw trash, which is different from our product.

However, we did find a person making the same prototype trash can as our project. A Japanese engineer called Minoru Kurata made a similar trash can as ours six years ago. His trash can was also able to catch the trash thrown by the user. The solution was similar to ours. There is a camera that captures the trash and a computer calculates the trajectory and sends the landing position to the trash can. The trash can then move to catch the trash. His trash can was custom made and the mobile mechanism was embedded in the trash can. The wheels that he used were regular wheels and the wheels rotated in line with the destination and then the trash can moves in a straight line to catch the trash. There is a special platform in the trash can that allows the wheels to rotate without the trash can rotating to reduce the overall motion. The engineer claimed that the success rate was about 20%. Compared to this prototype, our project has a higher success rate and uses mecanum wheels to move the trash can so that no rotational motion is needed.

VIII. Summary

Our system has met most of our specifications. The system is able to calculate the trajectory of the trash and the mobile platform was able to move to the landing point and catch the trash before it hits the ground. We had to modify the waiting position of the mobile platform from the corner to the center of the operation area in order to reach any point in time but the whole system performs well. One area of improvement was to improve the mounting system of the mecanum wheel and use better mecanum wheels so that the whole platform is not so sensitive to weight distribution and reduces the overall drift that affects the motion of the mobile platform.

A. *Future Work*

The main component that will really help excel in our project is improving the object detection part. We have tried two different methods for our object detection module: the GMG background subtraction and the customized YOLOv3 object detection model. Both methods have merits. If we can use the learning method to quickly detect and locate the ball and then use traditional tracking or motion capture methods to keep tracking the ball's movement, out whole vision pipeline can be both accurate and fast. In addition, our project can be more practical and general if a self-calibrated algorithm could be designed for the camera. Currently, users need to spend a considerate amount of time to calibrate the camera to make the 3d coordinates error under 5cm and it takes a long time to set up all the environments. This product can become much more practical and daily used if users can just click a button to self-calibrate the camera for a new environment. Lastly, it would be beneficial to self-design a more robust mobile platform so that it can have a stable center of mass, which provides equal forces for all four wheels.

B. *Lessons learned*

One key lesson learned from this project is to get the hardware components as soon as possible and also get

backup components as much as possible. If we had purchased the mobile platform earlier, we would be able to test it earlier and identify the drifting issue. At that time, it was possible for us to switch to other mobile platforms and better mecanum wheels. However, since in this project we did not do that, we have to stick with the hardware we had and come up with fixes that would alleviate the problem. Also, we had only purchased two embedded boards at first. When the first board short-circuited, we thought we supplied too much power to the board and hence caused the board to be burned. We did not realize that it was a short-circuit problem. When we burned the second board, we finally identified the problem, but it was too late. We had to purchase another board and it takes a week to ship since it was shipped from China. Although that did not affect our schedule as the following week was supposed to be coding work, it would still be better if we had more spare parts so that we won't need to buy and wait for a week for new spare parts.

Another one is that a real-time project is really challenging since there exists latency in every part of the project. It is very hard to optimize the code and the entire pipeline. One may need to use a better computer, better hardware, lower resolution pictures, simpler algorithms to make sure that the system can operate in real-time.

Last but not least, the mecanum wheel is very sensitive. The side movement requires equal forces and frictions from all four wheels. The drift of the center of the mass and the uneven floor material are all potential causes of incorrect movements of the mecanum wheel. It is important to design a mechanical structure that has a fixed center of mass and find an even floor material which provides enough friction.

REFERENCES

[1] Choi, Dooseop, et al. "Future Trajectory Prediction via RNN and Maximum Margin Inverse Reinforcement Learning." 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, doi:10.1109/icmla.2018.00026..

[2] Rozumnyi, Denys et al. "The World of Fast Moving Objects." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017): n. pag. Crossref. Web.

[3] Alberto Ruiz Garcıa, Denys et al. "Analisis de eventos deportivos mediante vision artificial." (2017).

[4] Instructables, & Makeblock Robotics. (2017, October 3). How to Make an All-direction Vehicle With Mecanum Wheels. Retrieved from https://www.instructables.com/id/All-direction-Vehicle-with-Mecanum-Wheels/.

[5] Minimal PyTorch implementation of YOLOv3, https://github.com/eriklindernoren/PyTorch-YOLOv3.

APPENDIX A. WORK SCHEDULE