In the following text,

1. computer vision
2. communication,
3. movement
4. controller

these aspects will be described.

Computer Vision

Dataset prepare

We take 500 multiple object images and label them by hands. In order to avoid overfitting or underfitting problems, we have made requirements for the lighting of the environment, the background and the placement of objects:

1. Green floor images:
    1. Multiple instances on the same images.
    2. Simulate real environment.
2. Mutative background images:
    1. Keep lighting strength changing.
    2. Background changing.
    3. No two same image.

There are 530 bottle object labels, 450 battery object labels, 320 cup object labels, 480 orange object labels and 280 paper object labels. From the testing result, the box and classification curve has converged.

Core computer vision algorithm Implementation

The CV framework is YOLOV5， which has the high performance in the low-power embedded devices. In the jetson nano, which has the 15 FPS (frames per second).

According to the data flow, we design the procedures are:

1. Base on the *YOLO V5* gets all objects in one video frame. It will classify all objects on this frame and get the left upper pixel position and right lower pixel position for each object. We use the small model because of the embedded performance limitation.

2. The *position detector* gets all depth of objects on one video frame. Base on the object pixel position on the frame, we use *realsense* to get depth value of this object. Too little detected depth value will cause fluctuations, which lead unstable in this system. For performance considerations, we choose 13 points and get the depth value of them. These 13 points form a "X" shape in the rectangular which is the banding box of each objects. Then remove the outline values which is not in the $[\mu - 3\sigma, \mu + 3\sigma]$ interval. Then get average value of rest of depth value as the final depth value of this object.

3. And all depth data pass the *depth comparator* to get the nearest object as the primary object, which the cart will get next.

4. The *primary object detector* gets the nearest object position. Base on the Carmera FOV (Field of View) and the resolution of camera, we can calculate the real position (x, y coordination and angle between head of cart and the object) in the 3D scenario. The x, y can be calculated by the below Python

code.

```
# x_pixels is the horizontal piexl number of camera
phi = math.atan2(n, x_pixels/2)
# depth is the depth value between primary object and cart.
x = depth * math.sin(phi)
y = depth * math.cos(phi)
```

5. Then take 20 results of *primary object detector* pass the *primary objects filter* to:

   ```
   1. remove the outline values,
   2. get average position,
   3. certain the class of object,
   ```

   This operation is used to avoid that any objects are not recognized in different frames, which can seriously affect the controller to make decisions.

6. Use the final result to generate movement commands. The commands will drive the 4 wheels to corresponding position.

Communication

The communication between Jetson nano and Respberry Pi base on TCP (Transmission Control Protocol) network connection. We made a Ethernet directly connect them without any router or switch. Jetson nano sends the position of the primary object to Raspberry Pi. It has a pack operation in Jetson nano side and unpack operation in Respberry Pi side.

Movement

According to the states:

1. last state, which include
    1. previous object class,
    2. previous cart position.
2. next object class.

The *movement controller* will calculate the movement track and generate commands. Then Raspberry Pi writes the commands to serial port, send to wheel motor. After parsing the base-16 commands, the motor perform corresponding movement.

Controller

It is used to coordinate the whole system run, load config file and control data flow.