ECE 470: Final Report
PIT STOP

Group: Pit stop

December 5, 2022

1 Team and Links

Group name: Pit stop
Group member:

- Yixuan Li, netid: yixuan19
- Yuhao Wang, netid: yuhaow7
- Weiang Wang, netid: weiangw2
- Yiming Li, netid: yiming20

Our group link is on github and youtube:

- github: https://github.com/StriderWYH/ECE470Project
- youtube:https://youtu.be/VVwhgkVU8xA

2 Introduction

With the development of science and technology, smart home has become more and more inseparable from people's lives. We always hope to have more reliable and convenient devices to greatly simplify our life. Have you ever been lying in bed, wanting to grab a drink or a snack, but not wanting to get out of bed? Or have you ever sat comfortably on the couch watching TV but refused to get up and grab some snacks?

Now we have a solution for that. With only two ur3 robots arms, one camera and one car, we can deliver food from kitchen to where you are.

And the approaches we use include OpenCV to recognize camera, inverse kinematics to guide the robot arm to the destination and the keyboard control for car to move.

Figure 1: home robot arm

3 Task description

Our whole task is picking some blocks(or anything you want robot to pick) from some place (like kitchen), delivering them to place you want (like you sofa) and puting the blocks there. The whole procedure can be divided into these several parts

- Firstly, the first ur3 robot arm picks the block and puts it on the car. At the same time, the second ur3 arm goes the position previously set, and the camera starts to see whether the target (car) has arrived.
- Secondly, the car then starts to move to the destination
- Thirdly, the car has arrived the destination, and camera will check whether the car is motionless (stop) and whether the target block has entered the workspace, if not, the terminal will raise warning to let the car move closer
- Finally, if the camera checks both are right, the second ur3 robot arm starts to move, given the position camera has detected, pick the block from the car, and put it on the dest place (maybe the owner's snack plate or bowl)

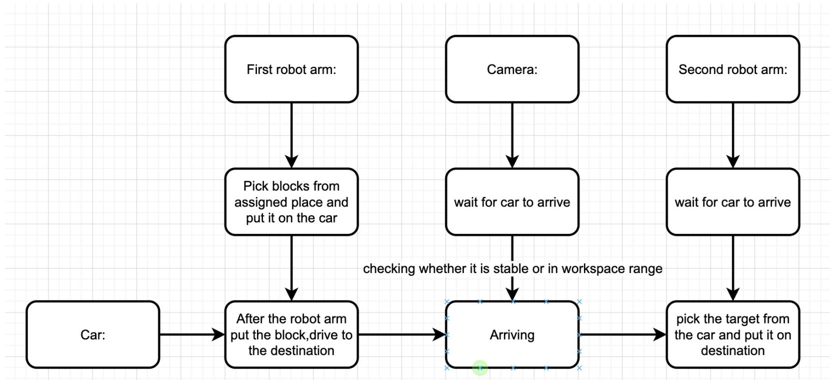
First robot arm:
Camera:
Second robot arm:
Pick blocks from assigned place and put it on the car
wait for car to arrive
wait for car to arrive
checking whether it is stable or in workspace range
Car:
After the robot arm put the block,drive to the destination
Arriving
pick the target from the car and put it on destination

Figure 2: The block diagram

3.1 Camera Setting

The camera is set to be on at the beginning, and then it will continuously check whether target has arrived, which is accomplished by checking the length of the array. If the target appears in the camera view, it will check two things

- Whether the target enters the workspace
- Whether the car is motionless.

Since the camera view is way bigger than workspace, so we need to check whether target is in a circle whose origin is arm, radius is 0.4.
When it comes to check the motion status, we compare the recognized coordinates of this scan and last scan, if they are the same, then start to move.

3.2 First step: pick and put from start position

This step is basically using the knowledge we learnt in lab2 and lab3. Since the start position is fixed, the coordinates is known. Therefor, we simply use inverse kinematics to calculate the angle. Additionally, we set middle position to ensure the arm will not collide with other stuff.

3.3 Second step: pick and put from start position

This step is basically using the knowledge we learnt in lab2 and lab3. Since the start position is fixed, the coordinates is known. Therefor, we simply use inverse kinematics to calculate the angle. Additionally, we set middle position to ensure the arm will not collide with other stuff.

3.4 Third step: move the car

In this project, we use a car that can move freely with keyboard control, whose file can be found in `"../src/drivers.'diff_wheeled_robot_gazebo/urdf'"`. We mainly use the keyboard to drive car to the destination. The car we use is based on half_A's codebase [1]

3.5 Final step: recognize target, pick and put

As we have explained, the camera will not give command to robot arm until the car is motionless and is in the workspace. Then we have the same thing as the lab5 did: calculate the world coordinate converted from camera view and use inverse kinematics to guide the robot arm to the destination and accomplish the task.

3.6 Detailed implement of inverse kinematics

3.6.1 The defined frame

Firstly, in inverse kinematics, we should prepare all the floowing that is needed to calculate the forward kinematics

- The end point coordinate that is specified by user: $(x_{wgrip}, y_{wgrip}, z_{wgrip})$.
- The Yaw angel that describes the angle end effector has moved: θ_{yaw}
- The calculated $\theta_0 - \theta_5$ to come up with the forward kinematic matrix

And for the above condition, we define $(x_{wgrip}, y_{wgrip}, z_{wgrip})$ to be the world coordinate which means left corner of the aluminum base plate which the UR3 is mounted on.

3.6.2 Derive local coordinate $(x_{grip}, y_{grip}, z_{grip})$

Establish the world coordinate frame (frame w) centered at the corner of the UR3's base Since We will solve the inverse kinematics problem in the base frame (frame 0) which the origin is centered in the base, but the provided coordinate is in the world fame which the origin is at the leftmost of the base, so we will immediately convert the coordinates entered by the user to base frame coordinates.

3.6.3 Derive wrist's center point $(x_{cen}, y_{cen}, z_{cen})$

Given the desired position of the gripper (xgrip, ygrip, zgrip) (in the base frame) and the yaw angle, we can get the center point coordinate by geometrical analysis.

The equation below is derived by us:

$$x_{cen} = x_{0grip} - L_9 * np.cos(yaw_{wgripdegree})$$
$$y_{cen} = y_{0grip} - L_9 * np.sin(yaw_{wgripdegree})$$
$$z_{cen} = z_{0grip}$$

3.6.4 Derive θ_1 using the previous value

Using the give length of the arm and the coordinate of center point, we can easily get the θ_1 as follows.

$$\theta_1 = \frac{\arcsin(-L_2 + L_4 - L_6)}{\sqrt{x_{cen}^2 + y_{cen}^2}} + \arctan(\frac{y_{cen}}{x_{cen}})$$

3.6.5 Derive θ_6 using the previous value

We can derive the θ_6 with the following formula:

$$\theta_6 = \theta_1 + \frac{\pi}{2} - \theta_{yaw}$$

3.6.6 Derive projected end point coordinate $(x_{3end}, y_{3end}, z_{3end})$

The derivation of the center point is as follows:

$$x_{3end} = x_{cen} + (L_4 + 27) * \sin\theta_1 - L_7 * \cos\theta_1$$

$$y_{3end} = y_{cen} - (L_4 + 27) * \cos\theta_1 - L_7 * \sin\theta_1$$

$$z_{3end} = z_{cen} + L_{10} + L_8$$

3.6.7 Derive the final three angles $\theta_2, \theta_3, \theta_4$

First for θ_2: we need to involve a middle value $d1$

$$d_1 = \sqrt{x_{3end}^2 + y_{3end}^2 + (z_{3end} - L_1)^2}$$

$$\alpha = \arcsin\frac{z_{3end} - L_1}{d1}$$

$$\theta_2 = -(\arccos\frac{\arccos({L_3}^2 + {d_1}^2 - {L_5}^2)}{2 * L_3 * d_1} + \alpha)$$

3.7 Choices that make things easier

To make the whole plan feasible, we make a lot of concessions.

- choose block instead of something closer related to reality like food or drinks in order to increase the accuracy of camera recognizing and picking.
- increase the max force and maxDistance of the gripper to make sure the pick is always successful.

4 Experimental Setup

We have roughly three main models in this part:

- car model
- ur3 robot
- simulation environment

4.1 Car model and ur3 robot

The model of the car is from halt_A's codebase [1]
And the robot arm we use is the ur3 robot arm, which is from the codebase which Ta gives.

4.2 Simulation environment

Since our project is aiming at delivering food and drinks to people in need home, we design a world that is in a house, with one kitchen and one living room. Our task is to deliver a drink (like coffee) in the kitchen to the people in the living room. Therefore, we introduce many models that is usual in the living room: writing desk, sofa, computer and so on. The models we use comes from a model base on a website [2]

4.3 Test

To test whether our code works, we firstly test our pick-and-put task in an empty world where the load is not so heavy. Since the first robot arm pick blocks at the given place, where zero error may occur, we mainly test the second arm which depends heavily on the camera.

To test whether the camera works well, we use a spawn function to generate blocks randomly within the workspace of the robot, to a max number of 12 blocks. In order to avoid regenerate blocks at the same place, we divided the workspace in to 80 points available to generate blocks based on the size of the block. Then we shuffle the array containing these points and pick location from it to generate blocks.

```
x_ranlist = [0,0,0,0]
    y_ranlist = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    for i in range(4):
        x_ranlist[i] = 0.1 + (i+1) * 0.035
    for j in range(18):
        y_ranlist[j] = -0.3 + (j+1) * 0.035

    random.shuffle(x_ranlist)
    random.shuffle(y_ranlist)
```

5 Data and Results

In the test part, we come up with two approaches to evaluate whether it is a success. Mostly on the pick and put task on second robot arm

- Binary discriminant standard:
  At first, we mainly test the result based on whether it can successfully suck up the block. That is, sucking up means success
- Coordinate Comparison:
  Then to make the whole procedure more accurate, we introduce the comparison of coordinates because it is much easier in gazebo to find the position of the block.

5.1 Data example

5.1.1 Binary discriminant standard

For this approach, we test the robot for 20 times, then count the success rounds.

Success	*Fail*
12	8

Table 1: This table shows the success and fail counts
Therefore, the success rate is 60%

5.1.2 Coordinate Comparison:

For the coordinate Comparison, we set a threshold for that method:

$$error\_dist = |real - calculated| = \sqrt{(x_r - x_{cal})^2 + (y_r - y_{cal})^2 + (z_r - z_{cal})^2}$$

Since the block size is 0.038, so we make the threshold to be 0.038/2 = 0.019
We altogether test 20 rounds, here are two examples. Here are two single examples of the method.

x_r	x_{cal}	y_r	y_{cal}	z_r	z_{cal}	$error\_dist$	$success$
0.231	0.257	0.114	0.135	0.038	0.038	0.033	0
0.142	0.171	0.212	0.243	0.038	0.038	0.042	0

Table 2: This table shows the two examples
According to our data, the success rate is 45%

5.2 Error analysis and solution

The error analysis:

- The parameters which are set on camera
- If the distance of camera is too far, the recognition also causes error

Solution:

- modify the parameters based on the accuracy
- make camera closer to the ground
- Increase the maxdistance of the gripper on the robot arm

5.3 Performance of robot arm

We can know from the graph that the initial accuracy is huge good, but after we use the second approach – that is raise the standard, the rate drops a lot. After we approach the solutions, but success rate grows a bit.

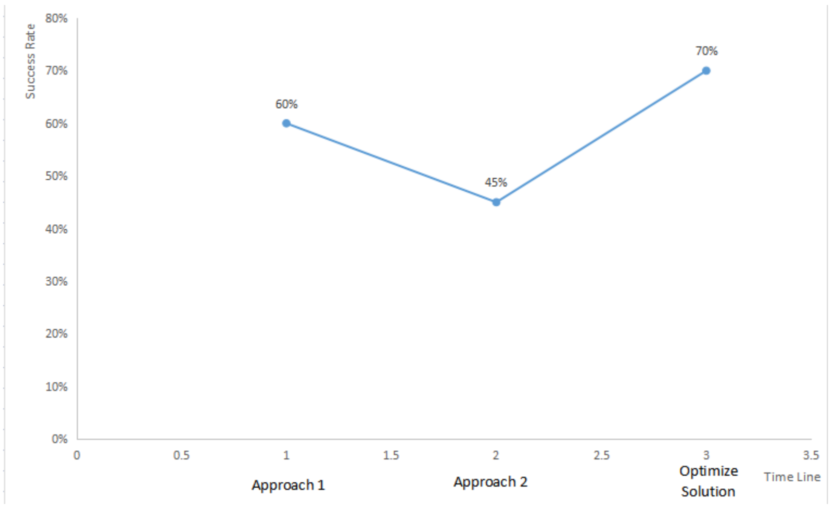

Figure 3: The performance of the robot

6 Summary and Challenges

6.1 What we have learnt

In this project, we have learnt:

- how to use the inverse kinetic to guide the robot to the position we want
- how to use the openCV to recognize the object and convert it into the corrdinate
- how to add a second robot to the world, know how to use group label; subscribers and publishers; generate msg files; how to launch a ROSService services
- how to add a robot with keyboard control
- how to use gazebo to simulate worlds, generate worlds and set constants

6.2 Challenges

For our group, the biggest challenge in this project is adding the second ur3 robot arm, which take tons of time and effort since the codebase TA gives us is used for one robot arm.

What we do, is Firstly, adding the model to the world, I use the "group" label to add the second robot in the namespace

Then, moving arm is a huge dilemma to us. we create a whole set of msg for the second arm in the MakefileList.txt. Then in the ur3_gazebo_driver, we copy a whole new set of callback functions for the second arm. In order to separate commands of two robots, we use different publishers and subscribers.

Till now, we can use different functions to move different arms. But only one of the grippers can work, then we modify the xacro file for the robot arms, and give different names to the grippers, then we can use ROSServices to call different grippers

6.3 Our plan

To be frank, we have changed our subject halfway since we find it is impossible to do, though it is a very interesting idea — change tire for the Formula Race Car.

Therefore, we waste a quite lot of time, with which we should come up with a more mature and complete project.

For this project, we find that the model we pick and put is still the block, which make simulation less like a reality. If given more time, we may change the model of it.

And we find that due to the limitation of the workspace, a lot of stuff what we want the robot to do end with fantasy. If given more time, we may try to do some make-up work, like adding a wheel to the robot.

References

[1] https://blog.csdn.net/weixin_44543463/article/details/120692649

[2] https://data.nvision2.eecs.yorku.ca/3DGEMS/